

## Supplementary Information S2. Code for differential VE and for the site-scanning methods.

```
# Primary analysis: genotype-specific VE

# The following R function sievecateg performs Prentice et al. (1978,
# Biometrics) and Lunn and McNeil (1995) Cox model analysis of genotype-
# specific VE
#
#
sievecateg <- function( flrtime, flrstatus, flrtype, Vx ) {

  # flrtime- right-censored failure time
  # flrstatus- indicator of observed infection (1=yes, 0=no)
  # flrtype- indicator of failure type (NA if not infected or infected and # failure type
not observed;
# 0 if infected and observed type 0; 1 if infected and observed # type 1)
# Vx- Vaccination status (0=placebo, 1=vaccine)

  # Standard Prentice et al. (1978) sieve analysis

  # flrtype = 0 or 1

  # Analysis of type 1:
  flrstatus0 <- flrstatus
  flrstatus0[flrstatus==1 & (is.na(flrtype) | flrtype==0)] <- 0
  fit <- coxph(Surv(flrtime,flrstatus0) ~ Vx)
  cat(paste("Type 1 VE results:","\n"))
  print(summary(fit))
  print(cox.zph(fit))

  # Analysis of type 0:
  flrstatus1 <- flrstatus
  flrstatus1[flrstatus==1 & (is.na(flrtype) | flrtype==1)] <- 0
  fit <- coxph(Surv(flrtime,flrstatus1) ~ Vx)
  cat(paste("Type 0 VE results:","\n"))
  print(summary(fit))
  print(cox.zph(fit))

  # Analysis of type 0:

  # Evaluate H0: VE(type 0) = VE(type 1) via the Lunn and McNeil (1995, Biometrics)
trick:

  currdelta <- flrstatus

  dblfuptime <- c(flrtime,flrtime)
  dblfustat <- c(currdelta,rep(0,length(currdelta)))
  dbltreatment <- c(Vx,Vx)
  firstdelta <- ifelse(currdelta==0 | (!is.na(flrtype) & flrtype==0),1,0)
```

```

seconddelta <- 1 - firstdelta
dbldelta <- c(firstdelta,seconddelta)
dblHIVinfection <- c(flirstatus,flirstatus)
dblflrtype <- c(flrtype,flrtype)

fit <- coxph(Surv(dblfuptime,dblufustat) ~ strata(dbldelta) + dbltreatment +
dbltreatment*dbldelta)

cat(paste("Differential VE results:","\n"))

print(summary(fit))

return()
} # sievecateg (..)

#####
# Carry out primary sieve analysis (this example shows the result calculation for just
sites 169 and 181, but the others are done in the same manner):

# The data variables must be set-up, for inputting into sievecateg:
# times: vector of failure times for all randomized subjects
#   (minimum of estimated infection time and right-censoring time)
# delta: indicator of HIV infected during the trial (1=yes, 0=no)
# gt169: indicator of whether the HIV infection was matched to the insert
#   at site 169 (NA if not infected)
# gt181: indicator of whether the HIV infection was matched to the insert
#   at site 181 (NA if not infected)
runDVE <- function ( times, delta, gt169, gt181 ) {

  cat(paste("results for genotype 169"),"\n")
  sievecateg(times,delta,gt169,Vx)

  cat(paste("results for genotype 181"),"\n")
  sievecateg(times,delta,gt181,Vx)

  # Repeat for joint genotypes

  # gt16918111      1,1 vs else
  # gt16918110     1,0 vs else
  # gt16918101     0,1 vs else
  # gt16918100     0,0 vs else

  gt16918111 <- ifelse(gt169==1 & gt181==1,1,0)
  gt16918110 <- ifelse(gt169==1 & gt181==0,1,0)
  gt16918101 <- ifelse(gt169==0 & gt181==1,1,0)
  gt16918100 <- ifelse(gt169==0 & gt181==0,1,0)

  cat(paste("results for genotype 169/181 1,1 vs else"),"\n")
  sievecateg(times,delta,gt16918111,Vx)

```

```

cat(paste("results for genotype 169/181 1,0 vs else"),"\n")
sievecateg(times,delta,gt16918110,Vx)

cat(paste("results for genotype 169/181 0,1 vs else"),"\n")
sievecateg(times,delta,gt16918101,Vx)

cat(paste("results for genotype 169/181 0,0 vs else"),"\n")
sievecateg(times,delta,gt16918100,Vx)

# Repeat for E vs. B infection
gtE <- ifelse(!is.na(gt169) | !is.na(gt181),1,0)
#sievecateg(times,delta,gtE,Vx)

# Kaplan-Meier type plots

delta169match <- ifelse(delta==1 & gt169==1,1,0)
delta169mismatch <- ifelse(delta==1 & gt169==0,1,0)
delta181match <- ifelse(delta==1 & gt181==1,1,0)
delta181mismatch <- ifelse(delta==1 & gt181==0,1,0)

postscript("cumincrcurvesby169181genotypes.ps")
par(mfrow=c(2,2),las=1,cex.axis=1.1,cex.lab=1.1,cex.main=1.3,oma=c(3,3,5,3))
KM <- survfit(Surv(times*(12/365), delta169match)~Vx, type="kaplan-meier")
plot(KM,fun="event",col=c("black","blue"),lty=c(1,3),xlab="Months since entry",
ylab="",lwd=1.7)
box()

legend(x="topleft",legend=c("Placebo","Vaccine"),col=c("black","blue"),lty=c(1,3),cex=1.1
,lwd=1.7)
title("Insert matched K169")

KM <- survfit(Surv(times*(12/365), delta169mismatch)~Vx, type="kaplan-meier")
plot(KM,fun="event",col=c("black","blue"),lty=c(1,3),xlab="Months since entry",
ylab="",lwd=1.7)
box()

legend(x="topleft",legend=c("Placebo","Vaccine"),col=c("black","blue"),lty=c(1,3),cex=1.1
,lwd=1.7)
title("Insert mismatched K169X")

KM <- survfit(Surv(times*(12/365), delta181match)~Vx, type="kaplan-meier")
plot(KM,fun="event",col=c("black","blue"),lty=c(1,3),xlab="Months since entry",
ylab="",lwd=1.7)
box()

legend(x="topleft",legend=c("Placebo","Vaccine"),col=c("black","blue"),lty=c(1,3),cex=1.1
,lwd=1.7)
title("Insert matched I181")

KM <- survfit(Surv(times*(12/365), delta181mismatch)~Vx, type="kaplan-meier")
plot(KM,fun="event",col=c("black","blue"),lty=c(1,3),xlab="Months since entry",

```

```

ylab="",lwd=1.7)
box()

legend(x="topleft",legend=c("Placebo","Vaccine"),col=c("black","blue"),lty=c(1,3),cex=1.1
,lwd=1.7)
title("Insert mismatched I181X")

mtext("Cumulative Probabilities of Genotype-Specific HIV
Infection",side=3,cex=1.4,outer=T,line=1)
dev.off()
} # runDVE(..)
####

#####

#####
# Secondary analyses: site-scanning methods

### Note: We use Benjamini-Hochberg (aka "fdr") instead of the qvalue package...
When the number of sites is small, it's hard to estimate pi0 in Storey's qvalue package.
It usually reverts to using pi0=1, which makes it equiv. to BH. Sometimes it just fails with
an error. It's safer to use p.adjust.
#library( "qvalue" ) # for "qvalue"

##### SETUP #####
# Set this to TRUE if you want to actually run the analyses when evaluating this file,
which requires that you have non-standard libraries "doMC" and "foreach". Otherwise,
you can set this to FALSE and manually run the scenarios using eg "run.scenario( 1 )".
run.when.sourced <- FALSE;

run.name.prefix <- "rv144";
## For sensitivity analysis to the second-to-be-infected subject in the linked transmission
pair, use this instead:
#run.name.prefix <- "rv144_noAA100";

sieve.methods <- c( "MBS", "GWJ", "SMMB" );
site.filter.sets <- c( "v1v2contactsites", "v1v2epimap" );
# NOTE: for focusing on v1v2, should also restrict genes to just "env".
genes <- c( "env" );
if( length( intersect( c( "v1v2contactsites", "v1v2epimap" ), site.filter.sets ) ) > 0 ) {
  stopifnot( all( genes == "env" ) );
}
env.nonv1v2.sites.filter.filename <-
"rv144/26Aug2011/mask/env.NonV1V2Columns.csv";
additional.site.filter.files <- list( all = c(), v1v2contactsites = c(
env.nonv1v2.sites.filter.filename,
"rv144/26Aug2011/mask/env.NonPeptideArrayHotspots.csv",
"rv144/26Aug2011/mask/env.NonAntibodyContactSites.csv" ), v1v2epimap = c(
env.nonv1v2.sites.filter.filename,
"rv144/26Aug2011/mask/env.NonV1V2EPIMAPContactSites.csv" ) );

```

```

gwj.weight.matrix.filename <- "~/src/from-svn/sieve/PAMamong.txt";
gwj.weight.matrix.is.probability.matrix <- TRUE;

```

```

insert.dir <- "rv144/26Aug2011/ref/individual_inserts/";
#####

```

```

runGWJ <- function ( vaccine.fasta.file.name, placebo.fasta.file.name,
insert.fasta.file.name, weight.matrix.file.name, weight.matrix.is.probability.matrix,
include.site, hxb2.map, run.name, output.file.name = paste( "gwj_", run.name, ".csv",
sep = "" ), q.value.significance.threshold = .2, num.perm.first.pass = 1000,
num.perm.refined = ( num.perm.first.pass * 100 ), p.value.refine.threshold = .15, ... )
{
  the.result <- tTestPermPvalWithRefinement( vaccine.fasta.file.name,
placebo.fasta.file.name, insert.fasta.file.name, weight.matrix.file.name,
weight.matrix.is.probability.matrix = weight.matrix.is.probability.matrix, include.site =
include.site, num.perm=num.perm.first.pass, num.perm.refined=num.perm.refined,
p.value.refine.threshold=p.value.refine.threshold, ... );
  the.result$hxb2.positions <- as.character( hxb2.map[ as.numeric( names(
the.result$p.values ) ), "hxb2Pos" ] )
  #.q.values <- qvalue( the.result$p.values[ !is.na( the.result$p.values ) ] )$qvalues;
  .q.values <- p.adjust( the.result$p.values[ !is.na( the.result$p.values ) ], method = "fdr"
);
  the.result$q.values <- the.result$p.values;
  the.result$q.values[ names( .q.values ) ] <- .q.values;
  result.table <- cbind( names( the.result$p.values ), the.result$hxb2.positions,
the.result$test.stats, the.result$p.values, the.result$q.values );
  rownames( result.table ) <- NULL;
  colnames( result.table ) <- c( "alignPos", "hxb2Pos", "t-stat", "p-value", "q-value" );
  write.csv( result.table, file=output.file.name, row.names = FALSE );
  ##### NOTE qvalue gives the same values as p.adjust( method="fdr" ) !!!
  #result.fdr.values <- p.adjust( result.p.values, method="fdr" );
  #all( abs( result.fdr.values - the.result$q.values ) < 1E-11 )
  if( any( the.result$q.values <= q.value.significance.threshold ) ) {
    return( result.table[ the.result$q.values <= q.value.significance.threshold, ] );
  } else {
    return( NA );
  }
}
} # runGWJ (..)

```

# NOTE: We have implemented the (Simplified) Mismatch Bootstrap method as an option in the GWJ code.

```

runSMMB <- function ( vaccine.fasta.file.name, placebo.fasta.file.name,
insert.fasta.file.name, include.site, hxb2.map, run.name, output.file.name = paste(
"smmb_", run.name, ".csv", sep = "" ), q.value.significance.threshold = .2,
num.perm.first.pass = 1000, num.perm.refined = ( num.perm.first.pass * 100 ),
p.value.refine.threshold = .15, ... )
{
  return( runGWJ( vaccine.fasta.file.name, placebo.fasta.file.name, insert.fasta.file.name,
weight.matrix.file.name=NULL, weight.matrix.is.probability.matrix=NA,

```

```

include.site=include.site,          hxb2.map=hxb2.map,          run.name=run.name,
output.file.name=output.file.name,
q.value.significance.threshold=q.value.significance.threshold,
num.perm.first.pass=num.perm.first.pass,          num.perm.refined=num.perm.refined,
p.value.refine.threshold=p.value.refine.threshold, mimic.smmb = TRUE, ... );
} # runSMMB (..)

```

```

runMBS <- function ( vaccine.fasta.file.name, placebo.fasta.file.name,
insert.fasta.file.name, include.site, hxb2.map, run.name, output.file.name = paste(
"mbs_", run.name, ".csv", sep = "" ), q.value.significance.threshold = .2,
num.perm.first.pass = 1000, num.perm.refined = ( num.perm.first.pass * 100 ),
p.value.refine.threshold = .15, prior.pseudocount = 1/21,
prior.expected.number.of.alt.sites = .5, return.posterior.S.modes = TRUE, ... )
{
  the.result <- MBS.withRefinement( vaccine.fasta.file.name=vaccine.fasta.file.name,
placebo.fasta.file.name=placebo.fasta.file.name,
insert.fasta.file.name=insert.fasta.file.name,          include.site=include.site,
prior.pseudocount=prior.pseudocount,
return.posterior.S.modes=return.posterior.S.modes,
prior.expected.number.of.alt.sites=prior.expected.number.of.alt.sites,
num.perm.first.pass=num.perm.first.pass,          num.perm.refined=num.perm.refined,
p.value.refine.threshold=p.value.refine.threshold, ... );
  the.result$hxb2.positions <- as.character( hxb2.map[ as.numeric( names(
the.result$p.values ) ), "hxb2Pos" ] );
  result.table <- cbind( names( the.result$p.values ), the.result$hxb2.positions,
the.result$posterior.alt.probs,          the.result$p.values,          the.result$q.values,
the.result$posterior.insertonly.probs, the.result$posterior.S.modes )
  rownames( result.table ) <- NULL;
  colnames( result.table ) <- c( "alignPos", "hxb2Pos", "posterior-alt-prob", "p-value", "q-
value", "posterior-insertonly-model-prob", "posterior-S-mode" );
  write.csv( result.table, file=output.file.name, row.names = FALSE );

  if( any( the.result$q.values <= q.value.significance.threshold ) ) {
    return( result.table[ the.result$q.values <= q.value.significance.threshold, ] );
  } else {
    return( NA );
  }
} # runMBS (..)

```

##### Utility functions #####

```

## Takes a file name of the PAM-style substitution probability matrix, with rows and
columns having AAs in alphabetical order (the last row/col is for the gap symbol). The
input file should have no headers and should be delimited by a single space.
## Returns the matrix, with rownames and colnames set appropriately.
readBetty <- function ( .file.name ) {
  .AA.chars <- c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S",
"T", "V", "W", "Y" );
  .gap.chars <- c( "-" );
  .AA.and.gap.chars <- c( .AA.chars, .gap.chars );
  Betty <- read.delim( .file.name, sep=" ", header=F );

```

```

rownames( Betty ) <- .AA.and.gap.chars;
colnames( Betty ) <- .AA.and.gap.chars;
## Sometimes the input file has 0s everywhere for gaps. If so, make it prob 1 to stay a
gap.
if( Betty[ "-", "-" ] == 0 ) {
  Betty[ "-", "-" ] <- 1;
}
return( as.matrix( Betty ) );
} ## readBetty( .. )

readPAM <- function ( .file.name ) {
  PAM.file <- file( .file.name, "r" );

  have.header <- FALSE;

  PAM.as.list <- list();

  line.text <- readLines( PAM.file, 1 );
  while( length( line.text ) > 0 ) {
    if( line.text == "" ) {
      # blank line. Do nothing but advance...
      line.text <- readLines( PAM.file, 1 );
      next;
    }
    if( length( grep( "^\\s*#", line.text ) ) > 0 ) {
      # comment line. Do nothing but advance...
      line.text <- readLines( PAM.file, 1 );
      next;
    }
    if( !have.header ) {
      # Then this should be the header line with the colnames.
      header <- unlist( strsplit( line.text, "\\s+" ) )[-1 ];
      have.header <- TRUE;
      line.text <- readLines( PAM.file, 1 );
      next;
    }
    line.data <- unlist( strsplit( line.text, "\\s+" ) );
    line.letter <- line.data[ 1 ];
    line.data <- as.numeric( line.data[-1 ] );
    names( line.data ) <- header;
    PAM.as.list[[ line.letter ]] <- line.data;

    # Now prepare for the next round.
    line.text <- readLines( PAM.file, 1 );
  } # End while( length( line.text ) > 0 )

  close( PAM.file );

  .AA.chars <- c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S",
"T", "V", "W", "Y" );
  .gap.chars <- c( "-" );

```

```

.AA.and.gap.chars <- c( .AA.chars, .gap.chars );

## The PAM header will have a stop codon symbol instead of a gap, but we'll use that
for the gap score.
PAM <- matrix( nrow=length( .AA.and.gap.chars ), ncol = length( .AA.and.gap.chars ) );
rownames( PAM ) <- .AA.and.gap.chars;
colnames( PAM ) <- .AA.and.gap.chars;
.stopToGap <- function ( .char ) { if( .char == "*" ) { return( "-" ); } else { return( .char ); }
};
.gapsToStops <- function ( .vector.of.chars ) { as.vector( sapply( .vector.of.chars,
function( .char ) { if( .char == "-" ) { return( "*" ); } else { return( .char ); } } ) ) );
lapply( names( PAM.as.list ), function( .original.char ) { .char <- .stopToGap(
.original.char ); if( .char %in% .AA.and.gap.chars ) { PAM[ .char, ] <- PAM.as.list[[
.original.char ]][.gapsToStops( .AA.and.gap.chars ) ]; } else { NA } } );

return( PAM );
} ## readPAM( .. )

### 'aaCountsOnePosition' returns frequencies of 'acceptable.chars' in the character
vector 'aa.char.vector'
aaCountsOnePosition <- function ( aa.char.vector, acceptable.chars = c( "A", "C", "D",
"E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ) )
{
return( sapply( acceptable.chars, function( .char ){ length( grep( .char, aa.char.vector ) )
} ) );
} ## aaCountsOnePosition(..)

### 'aaCountsPerPosition' returns a list with "aaCountsOnePosition" for each position.
aaCountsPerPosition <- function ( seq.envir, acceptable.chars = c( "A", "C", "D", "E", "F",
"G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ) )
{
seq.length <- nchar( get(ls(name=seq.envir)[1], env=seq.envir) );
return( lapply( 1:seq.length, function( .pos ) { return( aaCountsOnePosition(
aaPerPosition( seq.envir, .pos ), acceptable.chars=acceptable.chars ) ); } ) );
} # aaCountsPerPosition (..)

# Create the "include.site" vector for use in screenSites(..), based on a minimum-
variability criterion (by default, at least 3 sets have to have a seq with an AA the same as
the insert char at each included position, and at least 3 sets have to have a seq with an
AA that is different from that insert char). By default, one "set" is just one sequence, but
you can provide non-null values of vaccine.sequence.sets.list or
placebo.sequence.sets.list if you want to group sequences into sets (eg for multiple
sequences from one subject).
chooseSitesToScreen.minimumVariability <- function ( vaccine.seq.envir,
placebo.seq.envir, insert.seq.envir, vaccine.sequence.sets.list = NULL,
placebo.sequence.sets.list = NULL, min.mismatches.to.insert = 3, min.matches.to.insert
= min.mismatches.to.insert )
{
insert.char.vector <- strsplit(as.list(insert.seq.envir)[[1]],"")[1];

```



```

seq.length <- length( insert.char.vector );
include.site <- logical( length=seq.length );

if( is.null( vaccine.sequence.sets.list ) ) {
  vaccine.sequence.sets.list = lapply( ls( env=vaccine.seq.envir ), function( .seq.name )
{ .seq.name } );
  names( vaccine.sequence.sets.list ) <- ls( env=vaccine.seq.envir );
}
if( is.null( placebo.sequence.sets.list ) ) {
  stopifnot( !is.null( placebo.seq.envir ) );
  placebo.sequence.sets.list = lapply( ls( env=placebo.seq.envir ), function( .seq.name )
{ .seq.name } );
  names( placebo.sequence.sets.list ) <- ls( env=placebo.seq.envir );
}

vaccine.num.sequence.sets <- length( vaccine.sequence.sets.list );
placebo.num.sequence.sets <- length( placebo.sequence.sets.list );

for( i in 1:seq.length ){
  vaccine.chars.pos <- aaPerPosition( vaccine.seq.envir, i );
  placebo.chars.pos <- aaPerPosition( placebo.seq.envir, i );
  insert.char.pos <- insert.char.vector[ i ];

  # How many subjects have at least one match to the insert?
  matches.to.insert.pos.by.sequence.set <- sapply( vaccine.sequence.sets.list, function(
.sequences.in.set ) { as.numeric( any( vaccine.chars.pos[ .sequences.in.set ] ==
insert.char.pos, na.rm = TRUE ) ) } );
  matches.to.insert.pos <- sum( matches.to.insert.pos.by.sequence.set, na.rm=T );
  mismatches.to.insert.pos <- sum( ( 1 - matches.to.insert.pos.by.sequence.set ),
na.rm=T );

  if( matches.to.insert.pos < min.matches.to.insert ) {
    include.site[ i ] <- FALSE;
  } else if( mismatches.to.insert.pos < min.mismatches.to.insert ) {
    include.site[ i ] <- FALSE;
  } else {
    include.site[ i ] <- TRUE;
  }
}

return( include.site );
} # chooseSitesToScreen.minimumVariability (..)

### 'screenOutSites' returns an environment with sequences containing sites specified
in 'include.site'
# See also updateSiteSetsList(..)
screenOutSites <- function(seq.envir, include.site)
{
  stopifnot( is.environment( seq.envir ) );
  seq.length <- nchar(get(ls(name=seq.envir)[1], envir=seq.envir))

```

```

if (seq.length != length(include.site)){ stop(paste( "Sequence length and 'include.site'
vector length differ: seq.length=", seq.length, ", length(include.site)=",
length(include.site), ".", sep="" )) }
  for (seqid in ls(name=seq.envir)){
    seq <- get(seqid, env=seq.envir)
    seq <- paste(strsplit(seq,"")[[1]][include.site], collapse="")
    assign(seqid, seq, env=seq.envir)
  }
  return (seq.envir )
} ## screenOutSites(..)

```

# Since screenOutSites(..) actually removes sites from sequences, we might lose track of our numbering scheme. Eg what was site 50 before screening out sites 1-49 will become site 1 afterwards. This function takes a list of site-vectors, with each site falling in the range 1..length( include.site ). It returns the same list, but with the site values updated to the new numbering scheme after removing all sites for which include.site is FALSE. Note that sites that are not included will become NAs in the result list, so you might have site sets that are entirely NAs.

```

updateSiteSetsList <- function ( include.site, site.sets.list = NULL )
{
  if( is.null( site.sets.list ) ) {
    site.sets.list <- lapply( which( include.site ), function( .site ) { .site } );
  }
  old.site.to.new.site <- cumsum( as.numeric( include.site ) );
  old.site.to.new.site[ !include.site ] <- NA;
  return( lapply( site.sets.list, function( .site.set ) { old.site.to.new.site[ .site.set ] } ) );
} # updateSiteSetsList (..)

```

```

readFastaFile <- function ( .file.name, .acceptable.characters = c( "A", "C", "D", "E", "F",
"G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ),
.replace.unacceptable.characters = '-', .parent = parent.frame(), .seqid.sep = " ",
.seqid.length.limit = 1000, be.verbose = TRUE )
{
  if( be.verbose ) {
    cat( "Reading fasta file: " %+% .file.name, fill=TRUE );
  }

```

```

  return.hash <- new.env( hash=TRUE, parent=.parent );
  .fastaFile <- file( .file.name, "r" );

```

```

  .acceptable.characters.asonestring <- toupper( paste( .acceptable.characters, sep="",
collapse="" ) );

```

```

  .line.text <- readLines( .fastaFile, 1 );
  .seqid <- NULL;
  while( length( .line.text ) > 0 ) { # Outer loop looks for new sequence blocks (starting w
a seqid)
    if( !grep( "\\S", .line.text ) ) { # Just whitespace
      ## Skip whitespace lines.
      .line.text <- readLines( .fastaFile, 1 );
      next;

```

```

}
stopifnot( substr( .line.text, 1, 1 ) == ">" ); ## We expect sequence blocks to begin
with a description line
.seqid <- gsub( ">\\s*(\\S.*)" , "\\1" , .line.text ); ## Ignore ws between ">" and seqid.
.seqid <- substr( strsplit( .seqid, .seqid.sep )[[1]][1], 1, .seqid.length.limit ); # Take first
.seqid.length.limit chars up to first .seqid.sep char
if( exists( .seqid, inherits=TRUE, envir=return.hash ) ) { # If the .seqid already has an
associated sequence
warning( "Duplicate seqids (" %+% .seqid %+% "). Only the first instance will be
used!" );
.line.text <- readLines( .fastaFile, 1 );
next;
}
.line.text <- readLines( .fastaFile, 1 );
.temp.seq <- "";
while( length( .line.text ) > 0 ) { # Inner loop reads sequence associated with this block
if( substr( .line.text, 0, 1 ) == ">" ) { # Found the next block / seqid.
break; # Break from inner loop.
}
.temp.seq <- .temp.seq %+% .line.text;
.line.text <- readLines( .fastaFile, 1 );
} # End inner loop (reading in the lines of the current sequence)
if( is.na( .replace.unacceptable.characters ) ) {
stopifnot( grep( "[^" %+% .acceptable.characters.asonestring %+% "]" , .temp.seq,
invert=T ) );
} else {
.temp.seq <- gsub( "[^" %+% .acceptable.characters.asonestring %+% "]" ,
.replace.unacceptable.characters, .temp.seq );
}
assign( .seqid, .temp.seq, envir=return.hash );
} # End while( length( .line.text ) > 0 )

close( .fastaFile );
return( return.hash );
} # readFastaFile (..)

```

## Takes a list that maps seqid (strings) to aligned sequence (strings) and a filename, and optionally a prefix to be appended to the output sequences, and an optional first sequence number (if non-NA, a number will be prepended to the prefix and incremented for each sequence, to help ensure unique seqids).

## Returns nothing. Writes Fasta-formatted sequences as a side effect.

##

```

writeFastaFile <- function( .seq.list, .file.name, .seqid.prefix = "", .first.seqid.number = NA
)
{
.outFile <- file( .file.name, open="w" );
.seq.names <- names( .seq.list );
.seqid.number <- .first.seqid.number;
for( .seq.i in 1:length( .seq.list ) ) {
write( file=.outFile, ">" %+% ifelse( is.na( .seqid.number ), "", as.character(
.seqid.number ) %+% "|" ) %+% .seqid.prefix %+% .seq.names[ .seq.i ], append=T );
}
}

```

```

    if( !is.na( .seqid.number ) ) {
      .seqid.number <- .seqid.number + 1;
    }
    write( file=.outFile, .seq.list[.[.seq.i]], append=T );
  }
  close( .outFile );
} # writeFastaFile(..)

countFastaSeqs <- function ( seq.environment )
{
  return( length( as.list( seq.environment ) ) );
} ## countFastaSeqs (..)

### 'aaPerPosition' returns a character vector of aas at the position 'position' for
sequences in 'seq.environment'
## NOTE it'll return any char, including "unacceptable" chars.
aaPerPosition <- function ( seq.environment, position )
{
  return( unlist( sapply( as.list( seq.environment ), function( .seq ) { substr( .seq, position,
position ) } ) ) );
} ## aaPerPosition (..)

### charsToCategories returns a numeric vector of the indices of the given character
vector into the given acceptable.chars vector. Unacceptable chars will be replaced by
the given unacceptable.char.value (defaults to length( acceptable.chars ) + 1; meaning
that the actual number of categories may be greater than the number of acceptable
chars!
charsToCategories <- function ( char.vec, acceptable.chars = c( "A", "C", "D", "E", "F",
"G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ),
unacceptable.char.value = ( length( acceptable.chars ) + 1 ) )
{
  category.vec <- as.numeric( addNA( factor( char.vec, levels=acceptable.chars ) ) );
  if( is.na( unacceptable.char.value ) ) {
    unacceptable.char.value <- length( acceptable.chars ) + 1;
  }
  if( ( unacceptable.char.value != ( length( acceptable.chars ) + 1 ) ) && ( sum(
category.vec == ( length( acceptable.chars ) + 1 ) ) > 0 ) ) {
    category.vec[ category.vec == ( length( acceptable.chars ) + 1 ) ] <-
unacceptable.char.value;
  }
  return( category.vec );
} ## charsToCategories (..)

# This function takes a vector of values and returns a table with counts per categories,
but only counts values in the range 1:num.categories. Any value that isn't an integer or
isn't in that range will be ignored. Note that it's ok to pass NULL for
categorical.data.vector (it'll just be a table of 0s).
makeCountsTable <- function ( categorical.data.vector, num.categories )
{
  counts.table <- rep( 0, num.categories );
  names( counts.table ) <- 1:num.categories;

```

```

.tmp <- table( categorical.data.vector );
.tmp <- .tmp[ intersect( names( counts.table ), names( .tmp ) ) ];
counts.table[ names( .tmp ) ] <- .tmp;

return( counts.table );
} # makeCountsTable (..)

```

```
##### The sieve analysis methods #####
```

```
##### GWJ (and Simplified MMBootstrap) #####
```

```
#####
#####
```

```
### Description: Performs testing procedures to identify type A (t test)
```

```
### signature sites based on Gilbert, Wu, and Jobes (2008)
```

```
### Authors: Paul T Edlefsen, Michal Juraska, Peter Gilbert
```

```
#####
#####
```

```
### 'tStat' returns values of the t test statistic per aa position (or, if site.sets.list is non-NULL, one stat per site-set).
```

```
# If weight.matrix is NULL, weights will be 0 for insert-match and 1 for insert-mismatch.
```

```
# If site.sets.list is non-NULL, it should be a list of vectors of site numbers, with numbers referring to positions in the given seq.envirs. -- Note that these numbers must be in the range 1..nchar( get( ls( env=insert.seq.envir )[1], env=insert.seq.envir ) ), so if you've done any screening via eg screenOutSites(..), you'll need to make sure to update the values. See screenOutSites(..) and updateSiteSetsList(..).
```

```
# Note that by default the weights will be summed across the sites in a set, but you can change this behavior by changing the weights.across.sites.in.a.set.accumulation.fn -- but note that you should also change weights.across.sites.in.a.set.init from 0! This is the initial weight -- so for instance if you're using weights.across.sites.in.a.set.accumulation.fn = prod, you should set weights.across.sites.in.a.set.init to 1; if you're using weights.across.sites.in.a.set.accumulation.fn = min, you should set weights.across.sites.in.a.set.init to Inf; if you're using max, set weights.across.sites.in.a.set.init to -Inf.
```

```
tStat <- function( vaccine.seq.envir, placebo.seq.envir, insert.seq.envir, weight.matrix = NULL, site.sets.list = NULL, return.t.test.result = FALSE, weights.across.sites.in.a.set.init = 0, weights.across.sites.in.a.set.accumulation.fn = sum, vaccine.sequence.sets.list = NULL, placebo.sequence.sets.list = NULL, mimic.smb = FALSE, weights.across.sequences.in.a.set.accumulation.fn = if( mimic.smb ) { sum } else { mean }, acceptable.chars = ifelse( is.null( weight.matrix ), c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ), setdiff( rownames( weight.matrix ), 'X' ) ) )
```

```
{
insert.char.vector <- strsplit(as.list(insert.seq.envir)[[1]],"")[1];
if( is.null( site.sets.list ) ) {
site.sets.list = lapply( 1:length( insert.char.vector ), function( .site ) { .site } );
}
num.site.sets <- length( site.sets.list );

```

```

if( is.null( vaccine.sequence.sets.list ) ) {
  vaccine.sequence.sets.list = lapply( ls( env=vaccine.seq.envir ), function( .seq.name )
{ .seq.name } );
  names( vaccine.sequence.sets.list ) <- ls( env=vaccine.seq.envir );
}
if( is.null( placebo.sequence.sets.list ) ) {
  stopifnot( !is.null( placebo.seq.envir ) );
  placebo.sequence.sets.list = lapply( ls( env=placebo.seq.envir ), function( .seq.name )
{ .seq.name } );
  names( placebo.sequence.sets.list ) <- ls( env=placebo.seq.envir );
}

.vaccine.weights.by.pos <- list();
.placebo.weights.by.pos <- list();
if( mimic.smb ) {
  .vaccine.observed.seqs.count.by.pos <- list();
  .placebo.observed.seqs.count.by.pos <- list();
}
for( .pos.i in unique( unlist( site.sets.list ) ) ) {
  if( is.na( .pos.i ) ) {
    next;
  }
  insert.char.pos <- insert.char.vector[ .pos.i ];

  vaccine.chars.pos <- aaPerPosition( vaccine.seq.envir, .pos.i );
  vaccine.chars.pos[ !( vaccine.chars.pos %in% acceptable.chars ) ] <- NA;
  .seq.weights <- rep( NA, length( vaccine.chars.pos ) );
  names( .seq.weights ) <- names( vaccine.chars.pos );
  for( .seq.k in 1:length( .seq.weights ) ) {
    if( is.na( vaccine.chars.pos[ .seq.k ] ) ) {
      .weight <- NA;
    } else if( is.null( weight.matrix ) ) {
      # Use 0/1 indicators of insert mismatch (1 iff mismatch)
      .weight <- vaccine.chars.pos[ .seq.k ] != insert.char.pos;
    } else {
      .weight <- weight.matrix[ insert.char.pos, vaccine.chars.pos[ .seq.k ] ];
    }
    .seq.weights[ .seq.k ] <- .weight;
  } # End foreach vaccine seq.k
  .vaccine.weights.by.pos[[ .pos.i ]] <- sapply( vaccine.sequence.sets.list, function(
.sequences.in.set ) { weights.across.sequences.in.a.set.accumulation.fn( .seq.weights[
.sequences.in.set ], na.rm = T ) } );
  if( mimic.smb ) {
    .vaccine.observed.seqs.count.by.pos[[ .pos.i ]] <- sapply( vaccine.sequence.sets.list,
function( .sequences.in.set ) { sum( !is.na( .seq.weights[ .sequences.in.set ] ) ) } );
  }

  if( !is.null( placebo.seq.envir ) ) {
    placebo.chars.pos <- aaPerPosition( placebo.seq.envir, .pos.i );
    placebo.chars.pos[ !( placebo.chars.pos %in% acceptable.chars ) ] <- NA;
  }
}

```

```

.seq.weights <- rep( NA, length( placebo.chars.pos ) );
names( .seq.weights ) <- names( placebo.chars.pos );
for( .seq.k in 1:length( .seq.weights ) ) {
  if( is.na( placebo.chars.pos[ .seq.k ] ) ) {
    .weight <- NA;
  } else if( is.null( weight.matrix ) ) {
    # Use 0/1 indicators of insert mismatch (1 iff mismatch)
    .weight <- as.numeric( placebo.chars.pos[ .seq.k ] != insert.char.pos );
  } else {
    .weight <- weight.matrix[ insert.char.pos, placebo.chars.pos[ .seq.k ] ];
  }
  .seq.weights[ .seq.k ] <- .weight;
} # End foreach placebo seq.k
} # End if !is.null( placebo.seq.envir )
.placebo.weights.by.pos[[ .pos.i ]] <- sapply( placebo.sequence.sets.list, function(
.sequences.in.set ) { weights.across.sequences.in.a.set.accumulation.fn( .seq.weights[
.sequences.in.set ], na.rm = T ) } );
if( mimic.smmb ) {
  .placebo.observed.seqs.count.by.pos[[ .pos.i ]] <- sapply( placebo.sequence.sets.list,
function( .sequences.in.set ) { sum( !is.na( .seq.weights[ .sequences.in.set ] ) ) } );
}
} # End foreach .pos.i, calc weights..

test.statistics <- rep( NA, num.site.sets );
for( site.set.j in 1:num.site.sets ) {
  if( all( is.na( site.sets.list[[ site.set.j ] ] ) ) ) {
    # No sites in this set!
    next;
  }
  vaccine.weights <- rep( weights.across.sites.in.a.set.init, length(
vaccine.sequence.sets.list ) );
  placebo.weights <- rep( weights.across.sites.in.a.set.init, length(
placebo.sequence.sets.list ) );
  for( .pos.i in site.sets.list[[ site.set.j ] ] ) {
    if( is.na( .pos.i ) ) {
      # Skip NAs.
      next;
    }
    vaccine.weights <- apply( cbind( vaccine.weights, .vaccine.weights.by.pos[[ .pos.i ] ]
), 1, weights.across.sites.in.a.set.accumulation.fn );
    placebo.weights <- apply( cbind( placebo.weights, .placebo.weights.by.pos[[ .pos.i ] ]
), 1, weights.across.sites.in.a.set.accumulation.fn );
  } # End foreach pos .pos.i in site.set.j, update weights.

  if( return.t.test.result ) {
    if( all( vaccine.weights == vaccine.weights[ 1 ] ) && all( placebo.weights ==
vaccine.weights[ 1 ] ) ) {
      # Then there's no variation.
      test.statistics[ test.set.j ] <- NA;
    } else {
      t.test.result <- t.test( vaccine.weights, placebo.weights );
    }
  }
}

```

```

    test.statistics[ site.set.j ] <- list( t.test.result = t.test.result );
  }
} else {
  num.vaccine.weights <- sum( !is.na( vaccine.weights ) );
  num.placebo.weights <- sum( !is.na( placebo.weights ) );
  num.weights <- num.vaccine.weights + num.placebo.weights;
  if( mimic.smb ) {
    vaccine.observed.seqs.count <- sum( unlist( .vaccine.observed.seqs.count.by.pos[
site.sets.list[[ site.set.j ] ] ) ) );
    if( vaccine.observed.seqs.count > 0 ) {
      phat.v <- ( sum( vaccine.weights, na.rm=T ) / vaccine.observed.seqs.count );
    } else {
      phat.v <- 0;
    }
    placebo.observed.seqs.count <- sum( unlist( .placebo.observed.seqs.count.by.pos[
site.sets.list[[ site.set.j ] ] ) ) );
    if( placebo.observed.seqs.count > 0 ) {
      phat.p <- ( sum( placebo.weights, na.rm=T ) / placebo.observed.seqs.count );
    } else {
      phat.p <- 0;
    }
    test.statistics[ site.set.j ] <- ( phat.v - phat.p );
  } else {
    se <- sqrt( (num.vaccine.weights - 1)*var(vaccine.weights, na.rm=T)/(num.weights -
2) + (num.placebo.weights - 1)*var(placebo.weights, na.rm=T)/(num.weights - 2) );
    test.statistics[ site.set.j ] <- ( ( mean( vaccine.weights, na.rm=T ) - mean(
placebo.weights, na.rm=T ) ) / se );
  }
}

} # End foreach site.set.j
return( test.statistics );
} ## tStat (..)

```

### 'tPermTestStatistics' returns a matrix of t test statistics per aa position from 'num.perm' permuted data sets

```

# If weight.matrix is NULL, weights will be 0 for insert-match and 1 for insert-mismatch.
tPermTestStatistics <- function( vaccine.seq.env, placebo.seq.env, insert.seq.env,
weight.matrix = NULL, site.sets.list = NULL, weights.across.sites.in.a.set.init = 0,
weights.across.sites.in.a.set.accumulation.fn = sum, vaccine.sequence.sets.list = NULL,
placebo.sequence.sets.list = NULL, mimic.smb = FALSE,
weights.across.sequences.in.a.set.accumulation.fn = if( mimic.smb ) { sum } else {
mean }, num.perm = 100, use.bootstrap = ifelse( mimic.smb, TRUE, FALSE ),
be.verbose = FALSE, acceptable.chars = c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L",
"M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ) )
{
  if( is.null( site.sets.list ) ) {
    site.sets.list = lapply( 1:length( insert.char.vector ), function( .site ) { .site } );
  }
  num.site.sets <- length( site.sets.list );

```



```

combined.seq.env <- new.env( hash = TRUE );
lapply( ls( env=vaccine.seq.env ), function( .vaccine.sequence.name ) { assign(
.vaccine.sequence.name, get( .vaccine.sequence.name, env=vaccine.seq.env ),
env=combined.seq.env ) } );
lapply( ls( env=placebo.seq.env ), function( .placebo.sequence.name ) { assign(
.placebo.sequence.name, get( .placebo.sequence.name, env=placebo.seq.env ),
env=combined.seq.env ) } );

if( is.null( vaccine.sequence.sets.list ) ) {
vaccine.sequence.sets.list = lapply( ls( env=vaccine.seq.env ), function( .seq.name ) {
.seq.name } );
names( vaccine.sequence.sets.list ) <- ls( env=vaccine.seq.env );
}
if( is.null( placebo.sequence.sets.list ) ) {
placebo.sequence.sets.list = lapply( ls( env=placebo.seq.env ), function( .seq.name ) {
.seq.name } );
names( placebo.sequence.sets.list ) <- ls( env=placebo.seq.env );
}

.renamed.vaccine.sequence.sets.list <- vaccine.sequence.sets.list;
names( .renamed.vaccine.sequence.sets.list ) <- paste( "vaccine:", names(
vaccine.sequence.sets.list ), sep = "" );
.renamed.placebo.sequence.sets.list <- placebo.sequence.sets.list;
names( .renamed.placebo.sequence.sets.list ) <- paste( "placebo:", names(
placebo.sequence.sets.list ), sep = "" );

all.sequence.sets.list <- c( .renamed.vaccine.sequence.sets.list,
.renamed.placebo.sequence.sets.list );

vaccine.num.sequence.sets <- length( vaccine.sequence.sets.list );
placebo.num.sequence.sets <- length( placebo.sequence.sets.list );
num.sequence.sets <- vaccine.num.sequence.sets + placebo.num.sequence.sets;
stopifnot( num.sequence.sets == length( all.sequence.sets.list ) );

test.statistics.perm <- matrix( 0, nrow=num.site.sets, ncol=num.perm );
for( perm.i in 1:num.perm ) {
if( be.verbose && ( ( num.perm <= 100 ) || ( ( perm.i %% floor( num.perm / 100 ) ) == 0
) ) ) {
cat( paste( "Calling tStat(..) for", ifelse( use.bootstrap, "bootstrap", "permutation" ),
perm.i, "of", num.perm, "..\n" ) );
}
.permutation <- sample( 1:num.sequence.sets, replace = use.bootstrap );

.perm.vaccine.sequence.sets.list <- all.sequence.sets.list[ .permutation[
1:vaccine.num.sequence.sets ] ];
.perm.placebo.sequence.sets.list <- all.sequence.sets.list[ .permutation[ (
vaccine.num.sequence.sets + 1 ):num.sequence.sets ] ];
test.statistics.perm[ , perm.i ] <- tStat( combined.seq.env, NULL, insert.seq.env,
weight.matrix=weight.matrix, site.sets.list=site.sets.list,
weights.across.sites.in.a.set.init=weights.across.sites.in.a.set.init,
weights.across.sites.in.a.set.accumulation.fn=weights.across.sites.in.a.set.accumulation.

```

```

fn,                vaccine.sequence.sets.list=.perm.vaccine.sequence.sets.list,
placebo.sequence.sets.list=.perm.placebo.sequence.sets.list,
mimic.smmb=mimic.smmb,
weights.across.sequences.in.a.set.accumulation.fn=weights.across.sequences.in.a.set.a
ccumulation.fn, acceptable.chars=acceptable.chars );
} # End foreach permutation perm.i
return( test.statistics.perm );
} # tPermTestStatistics (..)

```

#### 'tTestPermPval' returns a list with the following components:

#### 'p.values' - numeric vector of unadjusted permutation-based t test p-values per aa position

# NOTE if include.site is "variable", then chooseSitesToScreen.minimumVariability(..) will be used (using the default values of min 3 insert-char and 3 non-insert-char seqs).

# NOTE: you can supply environments in place of filenames if you have already read in the fasta file(s).

# NOTE: If weight.matrix is NULL, weights will be 0 for insert-match and 1 for insert-mismatch.

# NOTE: by default each site is evaluated separately; change this behavior by specifying site sets (NULL means each site is in its own set). By default the weights will be summed across the sites in a set, but you can change this behavior by changing the weights.across.sites.in.a.set.accumulation.fn -- but note that you should also change weights.across.sites.in.a.set.init from 0! This is the initial weight -- so for instance if you're using weights.across.sites.in.a.set.accumulation.fn = prod, you should set weights.across.sites.in.a.set.init to 1; if you're using weights.across.sites.in.a.set.accumulation.fn = min, you should set weights.across.sites.in.a.set.init to Inf; if you're using max, set weights.across.sites.in.a.set.init to -Inf.

# NOTE: by default each sequence contributes a weight individually; change this behavior for eg multiple sequences-per-subject by specifying sequence sets (NULL means each sequence is in its own set). Note that permutation to determine the null will be by set (ie by subject), not by individual sequence. By default the weights will be averaged across the sequence in a set, but you can change this behavior by changing the weights.across.sequences.in.a.set.accumulation.fn, which should take a vector and should accept the "na.rm" argument (at least it shouldn't mind it; it'll always be called with na.rm = TRUE).

```

tTestPermPval <- function ( vaccine.fasta.file.name, placebo.fasta.file.name,
insert.fasta.file.name, weight.matrix.file.name = NULL, weight.matrix.is.probability.matrix
= FALSE, include.site = NULL, site.sets.list = NULL, weights.across.sites.in.a.set.init = 0,
weights.across.sites.in.a.set.accumulation.fn = sum, vaccine.sequence.sets.list = NULL,
placebo.sequence.sets.list = NULL, mimic.smmb = FALSE,
weights.across.sequences.in.a.set.accumulation.fn = if( mimic.smmb ) { sum } else {
mean }, num.perm = 1000, use.t.distribution.instead.of.permutation=( num.perm == 0 ),
use.bootstrap = ifelse( mimic.smmb, TRUE, FALSE ), be.verbose = TRUE,
acceptable.chars = c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R",
"S", "T", "V", "W", "Y", "-" ) )
{

```

```

  if( is.null( weight.matrix.file.name ) ) {
    # If weight.matrix is NULL, weights will be 0 for insert-match and 1 for insert-
mismatch.
    weight.matrix <- NULL;

```

```

if( be.verbose ) {
  cat( "Using mismatch indicators instead of weights.", fill = T );
}
} else {
if( weight.matrix.is.probability.matrix ) {
  # In probability form
  if( is.matrix( weight.matrix.file.name ) ) {
    weight.matrix <- 1 - weight.matrix.file.name; ## convert to a dissimilarity matrix
  } else {
    weight.matrix <- 1 - readBetty(weight.matrix.file.name); ## convert to a
dissimilarity matrix
  }
} else {
  # In log-likelihood-ratio (etc) form
  if( is.matrix( weight.matrix.file.name ) ) {
    weight.matrix <- 0 - weight.matrix.file.name; ## convert to a dissimilarity matrix
  } else {
    weight.matrix <- 0 - readPAM(weight.matrix.file.name); ## convert to a
dissimilarity matrix
  }
}
if( be.verbose ) {
  cat( "Using dissimilarity matrix converted from the", ifelse(
weight.matrix.is.probability.matrix, "probability-form substitution matrix", "substitution
matrix" ), "in file:", weight.matrix.file.name, fill = T );
}
} # End if is.null( weight.matrix.file.name ) .. else ..

if( is.environment( insert.fasta.file.name ) ) {
  insert.env <- insert.fasta.file.name;
} else {
  insert.env <- readFastaFile( insert.fasta.file.name, be.verbose=be.verbose );
}
if( is.environment( vaccine.fasta.file.name ) ) {
  vaccine.env <- vaccine.fasta.file.name;
} else {
  vaccine.env <- readFastaFile( vaccine.fasta.file.name, be.verbose=be.verbose );
}
if( is.environment( placebo.fasta.file.name ) ) {
  placebo.env <- placebo.fasta.file.name;
} else {
  placebo.env <- readFastaFile( placebo.fasta.file.name, be.verbose=be.verbose );
}

## TODO: ? .. For now, if include.site is _any_ character string, then use the
minimumVariability rules.
if( is.character( include.site ) ) {
  if( be.verbose ) {
    cat( "Calling chooseSitesToScreen.minimumVariability(..)", fill = T );
  }
}

```

```

include.site <- chooseSitesToScreen.minimumVariability( vaccine.env, placebo.env,
insert.env,
vaccine.sequence.sets.list=vaccine.sequence.sets.list,
placebo.sequence.sets.list=placebo.sequence.sets.list,
acceptable.chars=acceptable.chars );
}

if( !is.null( include.site ) ) {
insert.env <- screenOutSites( insert.env, include.site );
vaccine.env <- screenOutSites( vaccine.env, include.site );
placebo.env <- screenOutSites( placebo.env, include.site );
if( is.null( site.sets.list ) ) {
site.sets.list <- lapply( which( include.site ), function( .site ) { .site } );
names( site.sets.list ) <- which( include.site );
}
site.sets.list <- updateSiteSetsList( include.site, site.sets.list );
}

if( is.null( vaccine.sequence.sets.list ) ) {
vaccine.sequence.sets.list = lapply( ls( env=vaccine.env ), function( .seq.name ) {
.seq.name } );
names( vaccine.sequence.sets.list ) <- ls( env=vaccine.env );
} else {
# I haven't yet implemented the use.t.distribution.instead.of.permutation option for the
case where there's multiple seqs per subject.
stopifnot( !use.t.distribution.instead.of.permutation );
}
if( is.null( placebo.sequence.sets.list ) ) {
stopifnot( !is.null( placebo.env ) );
placebo.sequence.sets.list = lapply( ls( env=placebo.env ), function( .seq.name ) {
.seq.name } );
names( placebo.sequence.sets.list ) <- ls( env=placebo.env );
} else {
# I haven't yet implemented the use.t.distribution.instead.of.permutation option for the
case where there's multiple seqs per subject.
stopifnot( !use.t.distribution.instead.of.permutation );
}

if( use.t.distribution.instead.of.permutation ) {
pvals <- sapply( tStat( vaccine.env, placebo.env, insert.env, weight.matrix,
site.sets.list,
return.t.test.result=TRUE,
weights.across.sites.in.a.set.init=weights.across.sites.in.a.set.init,
weights.across.sites.in.a.set.accumulation.fn=weights.across.sites.in.a.set.accumulation.
fn, acceptable.chars=acceptable.chars ), function( .t.test.result ) { if( all( is.na(
.t.test.result ) ) ) { return( NA ); } else { return( .t.test.result$p.value ); } } );
} else {
if( be.verbose ) {
cat( "Calling tStat(..) for main test stats..\n" );
}
test.statistics <- tStat( vaccine.env, placebo.env, insert.env, weight.matrix,
site.sets.list=site.sets.list,
weights.across.sites.in.a.set.init=weights.across.sites.in.a.set.init,

```

```

weights.across.sites.in.a.set.accumulation.fn=weights.across.sites.in.a.set.accumulation.
fn,
vaccine.sequence.sets.list=vaccine.sequence.sets.list,
placebo.sequence.sets.list=placebo.sequence.sets.list,
mimic.smmb=mimic.smmb,
weights.across.sequences.in.a.set.accumulation.fn=weights.across.sequences.in.a.set.a
ccumulation.fn, acceptable.chars=acceptable.chars );
  if( be.verbose ) {
    cat( "Calling tStat(..) for", ifelse( use.bootstrap, "bootstraps", "permutations" ), "test
stats..\n" );
  }
  test.statistics.perm <- tPermTestStatistics( vaccine.env, placebo.env, insert.env,
weight.matrix,
num.perm=num.perm,
site.sets.list=site.sets.list,
weights.across.sites.in.a.set.init=weights.across.sites.in.a.set.init,
weights.across.sites.in.a.set.accumulation.fn=weights.across.sites.in.a.set.accumulation.
fn,
vaccine.sequence.sets.list=vaccine.sequence.sets.list,
placebo.sequence.sets.list=placebo.sequence.sets.list,
mimic.smmb=mimic.smmb,
weights.across.sequences.in.a.set.accumulation.fn=weights.across.sequences.in.a.set.a
ccumulation.fn,
use.bootstrap=use.bootstrap,
be.verbose=be.verbose,
acceptable.chars=acceptable.chars );

  calculatePValue <- function ( .pos.i, add.one.for.observed.test.statistic = FALSE ) { if(
is.na( test.statistics[ .pos.i ] ) ) { return( NA ); } else { return( min( 1, 2 * ( min( sum(
test.statistics[ .pos.i ] >= test.statistics.perm[ .pos.i, ], na.rm=T ), sum( test.statistics[
.pos.i ] <= test.statistics.perm[ .pos.i, ], na.rm=T ) ) + as.numeric(
add.one.for.observed.test.statistic ) ) / ( sum( !is.na( test.statistics.perm[ .pos.i, ] ) ) +
as.numeric( add.one.for.observed.test.statistic ) ), na.rm=T ) ); } };
  pvals <- sapply( 1:nrow( test.statistics.perm ), calculatePValue );
}

if( !is.null( site.sets.list ) ) {
  names( pvals ) <- names( site.sets.list );
}
out <- list( p.values=pvals, test.stats=test.statistics, include.site = include.site,
method=if( use.t.distribution.instead.of.permutation ) { "t.dist" } else if( use.bootstrap ) {
"bootstrap" } else { "permutation" } );
return( out );
} ## tTestPermPval (..)

tTestPermPvalWithRefinement <- function ( vaccine.fasta.file.name,
placebo.fasta.file.name, insert.fasta.file.name, weight.matrix.file.name = NULL,
include.site = NULL, num.perm = 1000, num.perm.refined = ( num.perm * 10 ),
p.value.refine.threshold = .15, be.verbose = TRUE, ... )
{
  vaccine.tempfile <- NULL;
  if( is.environment( vaccine.fasta.file.name ) ) {
    # Then it will be modified in the first call because of the call to screenOutSites(..).
    Let's just write it to a temporary file.
    vaccine.tempfile <- tempfile( fileext = ".fasta" );
    if( be.verbose ) {
      cat( "Saving the given vaccine sequences to a temporary file:", vaccine.tempfile, fill =
TRUE );
    }
  }
}

```

```

writeFastaFile( as.list( vaccine.fasta.file.name ), vaccine.tempfile );
}
placebo.tempfile <- NULL;
if( is.environment( placebo.fasta.file.name ) ) {
  # Then it will be modified in the first call because of the call to screenOutSites(..).
  Let's just write it to a temporary file.
  placebo.tempfile <- tempfile( fileext = ".fasta" );
  if( be.verbose ) {
    cat( "Saving the given placebo sequences to a temporary file:", placebo.tempfile, fill
= TRUE );
  }
  writeFastaFile( as.list( placebo.fasta.file.name ), placebo.tempfile );
}

the.result <- tTestPermPval( if( is.null( vaccine.tempfile ) ) { vaccine.fasta.file.name }
else { vaccine.tempfile }, if( is.null( placebo.tempfile ) ) { placebo.fasta.file.name } else {
placebo.tempfile },
weight.matrix.file.name=weight.matrix.file.name, insert.fasta.file.name,
num.perm=num.perm, be.verbose = be.verbose, ... );
include.site.refined <- the.result$include.site;
names( the.result$p.values ) <- which( include.site.refined );
include.site.refined[ as.numeric( names( the.result$p.values[ the.result$p.values >
p.value.refine.threshold ] ) ) ] <- FALSE;
if( ( num.perm.refined > 0 ) && any( include.site.refined ) ) {
  if( be.verbose ) {
    cat( "Refining ", sum( include.site.refined ), " positions: ", paste( which(
include.site.refined ), collapse = ", " ), sep = "", fill = TRUE );
  }
  the.result.refined <- tTestPermPval( if( is.null( vaccine.tempfile ) ) {
vaccine.fasta.file.name } else { vaccine.tempfile }, if( is.null( placebo.tempfile ) ) {
placebo.fasta.file.name } else { placebo.tempfile }, insert.fasta.file.name,
weight.matrix.file.name=weight.matrix.file.name, num.perm=num.perm.refined,
include.site=include.site.refined, be.verbose = be.verbose, ... );
  names( the.result.refined$p.values ) <- which( include.site.refined );
  the.result$p.values[ as.character( names( the.result.refined$p.values ) ) ] <-
the.result.refined$p.values;
}
if( !is.null( vaccine.tempfile ) ) {
  unlink( vaccine.tempfile );
}
if( !is.null( placebo.tempfile ) ) {
  unlink( placebo.tempfile );
}
return( the.result );
} # tTestPermPvalWithRefinement (..)

```

##### MBS #####

#####

#####

### Description: Evaluates the posterior probability and p-value of the variant of the Model-Based Sieve method tht is used for the RV144 v1v2 analysis.

```

### Author: Paul T Edlefsen
#####
#####
## NOTE: You'll need nonstandard package "caTools" if you want the posterior mean of
the isinsertonly parameter.

##### Main Interface and support fns #####

### This uses eg num.perm = 100 for the first pass, then refines any pvalues < .2 using
num.perm = 10000. Returns q-values < .2, and p-values and posterior.alt.probs for
those.
MBS.withRefinement <- function ( vaccine.fasta.file.name, placebo.fasta.file.name,
insert.fasta.file.name, prior.alt.prob = .5, prior.expected.number.of.alt.sites = NULL,
include.site = "variable", num.perm.first.pass = 100, num.perm.refined = (
num.perm.first.pass * 100 ), p.value.refine.threshold = .2, be.verbose = TRUE, ... )
{
  result      <-      MBS(      vaccine.fasta.file.name=vaccine.fasta.file.name,
placebo.fasta.file.name=placebo.fasta.file.name,
insert.fasta.file.name=insert.fasta.file.name,      prior.alt.prob=prior.alt.prob,
prior.expected.number.of.alt.sites=prior.expected.number.of.alt.sites,
include.site=include.site, num.perm=num.perm.first.pass, be.verbose=be.verbose, ... );

  include.site.refined <- result$include.site;
  include.site.refined[ as.numeric( names( result$p.values[ ( result$p.values >
p.value.refine.threshold ) ] ) ) ] <- FALSE;
  p.values <- result$p.values;

  if( ( num.perm.refined > 0 ) && any( include.site.refined ) ) {
    if( be.verbose ) {
      cat( "Refining ", sum( include.site.refined ), " positions: ", paste( which(
include.site.refined ), collapse = ", " ), sep = "", fill = TRUE );
    }
    result.refined <- MBS( vaccine.fasta.file.name=vaccine.fasta.file.name,
placebo.fasta.file.name=placebo.fasta.file.name,
insert.fasta.file.name=insert.fasta.file.name,      prior.alt.prob=result$prior.alt.prob,
prior.expected.number.of.alt.sites=NULL,      include.site=include.site.refined,
num.perm=num.perm.refined, be.verbose=be.verbose, ... );
    p.values[ as.character( names( result.refined$p.values ) ) ] <- result.refined$p.values;
  }
  if( length( p.values ) > 1 ) {
    ### Note: We use Benjamini-Hochberg (aka "fdr") instead of the qvalue package...
    When the number of sites is small, it's hard to estimate pi0 in Storey's qvalue package.
    It usually reverts to using pi0=1, which makes it equiv. to BH. Sometimes it just fails with
    an error.
    #library( "qvalue" );
    #q.values <- qvalue( p.values )$qvalues;
    q.values <- p.adjust( p.values, method = "fdr" );
  } else {
    q.values <- p.values;
  }
}

```

```

result$p.values <- p.values;
result$q.values <- q.values;

return( result );
} # MBS.withRefinement (..)

# NOTE if include.site is "variable", then chooseSitesToScreen.minimumVariability(..)
will be used (using the default values of min 3 insert-char and 3 non-insert-char seqs).
# NOTE: you can supply environments in place of filenames if you have already read in
the fasta file(s).
MBS <- function ( vaccine.fasta.file.name, placebo.fasta.file.name, insert.fasta.file.name,
acceptable.chars = c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R",
"S", "T", "V", "W", "Y", "-" ), include.site = NULL, num.perm = 100,
prior.expected.number.of.alt.sites = NULL, prior.alt.prob = ifelse( is.null(
prior.expected.number.of.alt.sites ), .5, NA ), alternative = "two.sided",
return.posterior.S.modes = FALSE, return.posterior.S.means = FALSE,
return.posterior.S.medians = FALSE, be.verbose = TRUE, ... )
{
  stopifnot( ( alternative == "two.sided" ) || ( alternative == "greater" ) || ( alternative ==
"less" ) );
  # A rather complex way to calculate a p-value, but there it is.
  if( alternative == "two.sided" ) {
    calculatePValue <- function ( .stat, .perm.stats, add.one.for.observed.test.statistic =
FALSE ) { if( is.na( .stat ) ) { return( NA ); } else { return( min( 1, 2 * ( min( sum( .stat >=
.perm.stats, na.rm=T ), sum( .stat <= .perm.stats, na.rm=T ) ) + as.numeric(
add.one.for.observed.test.statistic ) ) / ( sum( lis.na( .perm.stats ) ) + as.numeric(
add.one.for.observed.test.statistic ) ), na.rm=T ) ); } };
  } else if( alternative == "greater" ) {
    calculatePValue <- function ( .stat, .perm.stats, add.one.for.observed.test.statistic =
FALSE ) { if( is.na( .stat ) ) { return( NA ); } else { return( ( sum( .stat <= .perm.stats,
na.rm=T ) + as.numeric( add.one.for.observed.test.statistic ) ) / ( sum( lis.na(
.perm.stats ) ) + as.numeric( add.one.for.observed.test.statistic ) ) ); } };
  } else { # alternative == "less"
    calculatePValue <- function ( .stat, .perm.stats, add.one.for.observed.test.statistic =
FALSE ) { if( is.na( .stat ) ) { return( NA ); } else { return( ( sum( .stat >= .perm.stats,
na.rm=T ) + as.numeric( add.one.for.observed.test.statistic ) ) / ( sum( lis.na(
.perm.stats ) ) + as.numeric( add.one.for.observed.test.statistic ) ) ); } };
  }

  if( is.environment( insert.fasta.file.name ) ) {
    insert.env <- insert.fasta.file.name;
  } else {
    insert.env <- readFastaFile( insert.fasta.file.name, be.verbose=be.verbose );
  }
  if( is.environment( vaccine.fasta.file.name ) ) {
    vaccine.env <- vaccine.fasta.file.name;
  } else {
    vaccine.env <- readFastaFile( vaccine.fasta.file.name, be.verbose=be.verbose );
  }
  if( is.environment( placebo.fasta.file.name ) ) {

```



```

    placebo.env <- placebo.fasta.file.name;
  } else {
    placebo.env <- readFastaFile( placebo.fasta.file.name, be.verbose=be.verbose );
  }

  ## TODO: ? .. For now, if include.site is _any_ character string, then use the
  minimumVariability rules.
  if( is.character( include.site ) ) {
    if( be.verbose ) {
      print( "Calling chooseSitesToScreen.minimumVariability(..)" );
    }
    include.site <- chooseSitesToScreen.minimumVariability( vaccine.env, placebo.env,
insert.env );
    if( be.verbose ) {
      print( paste( "Including ", sum( include.site ), " sites", sep="" ) );
    }
  }

  if( !is.null( include.site ) ) {
    insert.env <- screenOutSites( insert.env, include.site );
    vaccine.env <- screenOutSites( vaccine.env, include.site );
    placebo.env <- screenOutSites( placebo.env, include.site );
  } # End if !is.null( include.site )

  if( is.null( include.site ) ) {
    num.sites <- nchar( get( ls( env=insert.env )[ 1 ], env=insert.env ) );
    position.names <- 1:num.sites;
  } else {
    position.names <- which( include.site );
  }

  if( !is.null( prior.expected.number.of.alt.sites ) ) {
    prior.alt.prob <- min( 1, prior.expected.number.of.alt.sites / length( position.names ) );
    if( be.verbose ) {
      cat( paste( "Setting prior.alt.prob to", prior.alt.prob, "to get a prior expected number
of sieved sites of", prior.expected.number.of.alt.sites, "(we are testing", length(
position.names ), "sites).\n" ) );
    }
    if( prior.alt.prob == 1 ) {
      ## TODO: ?
      warn( "Prior alt prob is 1!" );
    }
  }

  test.statistics <- MBS.multiplePositions( vaccine.env, placebo.env, insert.env,
acceptable.chars=acceptable.chars, prior.alt.prob = prior.alt.prob,
calculate.p.value.num.permutations = num.perm, position.names = position.names,
return.S.posterior.mode = return.posterior.S.modes, return.S.posterior.mean =
return.posterior.S.means, return.S.posterior.median = return.posterior.S.medians,
calculatePValue.fn = calculatePValue, be.verbose=be.verbose, ... );

```

```

posterior.alt.probs <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.alt.prob } );
posterior.insertonly.probs <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.insertonly.prob } );

return.list <- list( alternative = alternative, prior.alt.prob = prior.alt.prob, include.site =
include.site, posterior.alt.probs = posterior.alt.probs, posterior.insertonly.probs =
posterior.insertonly.probs );

if( num.perm > 0 ) {
return.list$p.values <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.alt.prob.p.value } );
}

if( return.posterior.S.modes ) {
return.list$posterior.S.modes <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.S.mode } );
} # End if return.posterior.S.modes
if( return.posterior.S.means ) {
return.list$posterior.S.means <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.S.mean } );
}
if( return.posterior.S.medians ) {
return.list$posterior.S.medians <- sapply( test.statistics, function( .MBS.result ) {
.MBS.result$posterior.S.median } );
}

return( return.list )
} ## MBS (..)

### 'MBS.multiplePositions' returns values of the test statistics per aa position
MBS.multiplePositions <- function ( vaccine.seq.envir, placebo.seq.envir,
insert.seq.envir, acceptable.chars = c( "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M",
"N", "P", "Q", "R", "S", "T", "V", "W", "Y", "-" ), prior.pseudocount = 1/length(
acceptable.chars ), position.names = NULL, be.verbose = TRUE, ... )
{
insert.char.vector <- strsplit(as.list(insert.seq.envir)[[1]],"")[1];
seq.length <- length( insert.char.vector );

vaccine.num.seqs <- countFastaSeqs( vaccine.seq.envir );
placebo.num.seqs <- countFastaSeqs( placebo.seq.envir );
num.seqs <- vaccine.num.seqs + placebo.num.seqs;

test.statistics <- list();

prior.alphas.local <- rep( prior.pseudocount, length( acceptable.chars ) );
for( i in 1:seq.length ) {
vac.data.pos <- charsToCategories( aaPerPosition( vaccine.seq.envir, i ),
acceptable.chars = acceptable.chars, unacceptable.char.value = NA );
pla.data.pos <- charsToCategories( aaPerPosition( placebo.seq.envir, i ),
acceptable.chars = acceptable.chars, unacceptable.char.value = NA );
}
}

```

```

insert.category.pos <- charsToCategories( aaPerPosition( insert.seq.envir, i ),
acceptable.chars = acceptable.chars, unacceptable.char.value = NA );
add.pla.count.data.to.prior.alphas.pos <- FALSE;

pla.count.data.pos <- makeCountsTable( pla.data.pos, length( acceptable.chars ) );

if( be.verbose ) {
  if( is.null( position.names ) ) {
    cat( "pos", i, "\n" );
  } else {
    cat( "pos", position.names[ i ], "\n" );
  }
} # End if be.verbose

test.statistics[[ i ]] <- MBS.oneposition( vac.data.pos, calculate.p.value.pla.data =
pla.data.pos, insert.category = insert.category.pos, prior.alphas = prior.alphas.local,
add.pla.count.data.to.prior.alphas = pla.count.data.pos, be.verbose=be.verbose, ... );
} # End foreach i in 1:seq.length

if( !is.null( position.names ) ) {
  names( test.statistics ) <- position.names;
}
return( test.statistics );
} # MBS.multiplePositions (..)

```

# NOTE: Use prior.alt.prob to set the prior probability of the alternative model to something other than .5.

# NOTE: If you want to calculate a p-value, you need to specify the number of permutations to conduct. The permutations are of the class labels (pla and vac) so if pla.data is NULL, you must specify the placebo data using calculate.p.value.pla.data.

# NOTE: add.pla.count.data.to.prior.alphas should be the counts from calculate.p.value.pla.data. We parameterize it this way to facilitate using the randomly-permuted pla data to augment the prior.alphas when this is called recursively.

# NOTE: noninsert-model reallocation of sieved mass (for the non-isinsertonly case) will be uniform over noninsert AAs.

# NOTE: By default, calculate.p.value.num.permutations is NA, so no p-value will be calculated. If you set it to a number > 0, this will calculate a one-sided p-value as the fraction of permutations with LLRs >= the observed LLR -- unless you provide an alternative function for the "calculatePValue.fn". That's how you'd get a "two-sided" p-value, for instance. The fn's signature should be calculatePValue.fn( .stat, .permutation.stats ).

# NOTE: if calculate.p.value.num.permutations is a matrix, it should be a matrix with as many rows as desired permutations, each row being a permutation of the indices of all of the data-to-be-permuted (eg permutations of 1:(length(vac.data)+length(calculate.p.value.pla.data))). The first (num.placebo.subjects) cols of any given row will be used as the indices into the original data (concatenated placebo, then vaccine data) for that permutation's placebo data, the rest for the vac data.

```

MBS.oneposition <- function ( vac.data, calculate.p.value.pla.data, insert.category = 1,
prior.alphas, add.pla.count.data.to.prior.alphas = FALSE, prior.alt.prob = .5,

```

```

insertonlymodel.prob = .5, calculate.p.value.num.permutations = NA,
return.S.posterior.mode = FALSE, S.mode.tolerance = 1E-3, return.S.posterior.mean =
FALSE, return.S.posterior.median = FALSE, calculatePValue.fn = NULL, be.verbose =
TRUE )
{
  if( !is.na( calculate.p.value.num.permutations ) && ( calculate.p.value.num.permutations
> 0 ) ) {
    # You can't do the permutations if there's nothing to permute..
    stopifnot( !is.null( calculate.p.value.pla.data ) );
  }

  n.pla <- length( calculate.p.value.pla.data );
  n.vac <- length( vac.data );
  n.subjects <- n.pla + n.vac;
  if( is.matrix( calculate.p.value.num.permutations ) ) {
    calculate.p.value.permutation.indices <- calculate.p.value.num.permutations;
    calculate.p.value.num.permutations <- nrow( calculate.p.value.permutation.indices );
  } else if( !is.na( calculate.p.value.num.permutations ) ) {
    if( calculate.p.value.num.permutations == 0 ) {
      calculate.p.value.num.permutations <- NA;
    } else {
      calculate.p.value.permutation.indices <- matrix( nrow =
calculate.p.value.num.permutations, ncol = n.subjects );

      for( .permutation.i in 1:calculate.p.value.num.permutations ) {
        calculate.p.value.permutation.indices[ .permutation.i, ] <- sample.int( n.subjects,
n.subjects, replace=F );
      }
    }
  } # End if is.matrix( calculate.p.value.permutation.indices ) .. else if !is.na(..)

  num.categories <- length( prior.alphas );

  prior.alphas.orig <- prior.alphas;
  prior.alphas <- prior.alphas + add.pla.count.data.to.prior.alphas;

  .MBS.oneposition.helper <- function ( .vac.data, .prior.alphas, .return.S.posterior.mean
= FALSE, .return.S.posterior.median = FALSE )
  {
    return.list <- list();

    vac.count.data <- makeCountsTable( .vac.data, num.categories );

    return.list$log.null.prob <- dmultinom( vac.count.data, prob=( .prior.alphas / sum(
.prior.alphas ) ), log = TRUE );

    .log.alternative.prob.result <- MBS.likelihood.partialrealloc.numerical.integration(
.prior.alphas, vac.count.data, alphaS = c( 1, 1 ), log.results = TRUE,
insertonlymodel.prob=insertonlymodel.prob, insert.category=insert.category,
return.isinsertonly.posterior = TRUE, return.S.posterior.mode=return.S.posterior.mode,
S.mode.tolerance = S.mode.tolerance,

```

```

return.S.posterior.mean=.return.S.posterior.mean,
return.S.posterior.median=.return.S.posterior.median );
  return.list$log.alternative.prob <- .log.alternative.prob.result$density;
  return.list$posterior.insertonly.prob <-
.log.alternative.prob.result$insertonly.posterior.prob;
  if( return.S.posterior.mode ) {
    return.list$posterior.S.mode <- .log.alternative.prob.result$S.mode;
  }
  if( .return.S.posterior.mean ) {
    return.list$posterior.S.mean <- .log.alternative.prob.result$S.mean;
  }
  if( .return.S.posterior.median ) {
    return.list$posterior.S.median <- .log.alternative.prob.result$S.median;
  }

  posterior.alt.prob.num <- ( prior.alt.prob * exp( return.list$log.alternative.prob ) );
  return.list$posterior.alt.prob <- posterior.alt.prob.num / ( posterior.alt.prob.num + ( ( 1 -
prior.alt.prob ) * exp( return.list$log.null.prob ) ) );

  if( return.S.posterior.mode ) {
    return.list$log.alternative.prob.at.posterior.S.mode.minus.log.null.prob <- (
return.list$log.alternative.prob.at.posterior.S.mode - return.list$log.null.prob );
  }

  return( return.list );
} # .MBS.oneposition.helper (..)

augmented.return.list <- .MBS.oneposition.helper( vac.data, prior.alphas,
.return.S.posterior.mean = return.S.posterior.mean, .return.S.posterior.median =
return.S.posterior.median );

if( !is.na( calculate.p.value.num.permutations ) && ( calculate.p.value.num.permutations
> 0 ) ) {
  # This is a redundant test (we test it above, too):
  # Make sure there's some pla.data for permuting.
  stopifnot( !is.null( calculate.p.value.pla.data ) );
  if( is.null( calculatePValue.fn ) ) {
    # Then use the "greater" one-side p-value.
    calculatePValue.fn <- function( .stat, .permutation.stats ) { if( is.na( .stat ) ) { return(
NA ); } else { return( sum( ( .permutation.stats >= .stat ), na.rm = TRUE ) / sum( !is.na(
.permutation.stats ) ) ); } }
  }

  data.for.permutation <- c( calculate.p.value.pla.data, vac.data );

  augmented.return.list$permutation.posterior.alt.probs <- rep( NA,
calculate.p.value.num.permutations );
  for( .permutation.i in 1:calculate.p.value.num.permutations ) {
    .permutation.pla.data <- data.for.permutation[ calculate.p.value.permutation.indices[
.permutation.i, 1:n.pla ] ];

```

```

    .permutation.vac.data <- data.for.permutation[ calculate.p.value.permutation.indices[
.permutation.i, ( n.pla+1 ):( n.subjects ) ] ];
    .permutation.prior.alphas <- prior.alphas.orig;
    if( !is.logical( add.pla.count.data.to.prior.alphas ) || add.pla.count.data.to.prior.alphas
) {
    .permutation.pla.count.data <- makeCountsTable( .permutation.pla.data,
num.categories );
    .permutation.prior.alphas <- prior.alphas.orig + .permutation.pla.count.data;
    }

    # Note that we don't bother calculating posteriors of S for the permuted data (except
the S mode, which we need to calculate the at.posterior.S.mode p-values).
    # Note that we've already added the (permutation-specific) pla.count data to the
prior.alphas.
    .MBS.oneposition.helper.result <-
    .MBS.oneposition.helper( .vac.data = .permutation.vac.data, .prior.alphas =
.permutation.prior.alphas, .return.S.posterior.mean = FALSE, .return.S.posterior.median
= FALSE );
    if( return.S.posterior.mode ) {
    }
    augmented.return.list$permutation.posterior.alt.probs[ .permutation.i ] <-
.MBS.oneposition.helper.result$posterior.alt.prob;
    } # End foreach .permutation.i

    augmented.return.list$posterior.alt.prob.p.value <- calculatePValue.fn(
augmented.return.list$posterior.alt.prob,
augmented.return.list$permutation.posterior.alt.probs );
    } # End if we should calculate a p-value.

    return( augmented.return.list );
} # MBS.oneposition (..)

# If all of return.S.posterior.* are FALSE (the default), the return value is a density. If
return.isinsertonly.posterior or any return.S.posterior.* is TRUE, the return value is a list
(with elements "density" and one or more of "isinsertonly", "S.mode", "S.mean",
"S.median").
MBS.likelihood.partialrealloc.numerical.integration <- function ( prior.alphas, vac.counts,
alphaS = c( 1, 1 ), log.results = FALSE, insertonlymodel.prob = .5, insert.category = 1,
return.isinsertonly.posterior = FALSE, return.S.posterior.mode = FALSE,
S.mode.tolerance = 1E-3, return.S.posterior.mean = FALSE, return.S.posterior.median =
FALSE )
{
if( return.isinsertonly.posterior ) {
library( "caTools" ); # for trapz()
}

pla.counts.ins <- NULL;
pla.counts.nonins <- NULL;

vac.counts.ins <- c( vac.counts[ insert.category ], sum( vac.counts[ -insert.category ] ) );
vac.counts.nonins <- vac.counts[ -insert.category ];

```

```

alpha.ins <- c( prior.alphas[ insert.category ], sum( prior.alphas[ -insert.category ] ) );
alpha.nonins <- prior.alphas[ -insert.category ];

# Normalize so we can use these as conditional probabilities of emitting each noninsert
symbol (given that it's a noninsert symbol).
alpha.nonins <- alpha.nonins / sum( alpha.nonins );

## First, the insertonlypart of the nonins likelihood, which doesn't depend on S. Note
that the pla part is the same for insertonly and noninsertonly.
r0.nonins.insertonlypart <- dmultinom( vac.counts.nonins, prob=alpha.nonins, log =
FALSE );

# This is for saving the values of S, for getting posteriors thereof (marginal and by
isinsertonly).
.evaluated.Ss.and.values <- matrix( ncol=4, nrow=0 );
# NOTE: As a side effect, calling this modifies .evaluated.Ss.and.values:
MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS <- function (
evaluate.atS )
{
# Note evaluate.atS will be a vector (when called via integrate(..)).

## Note that what this fn expects is the reallocated mass as a fraction of the total
noninsert mass, so it's  $S \cdot p / (1 - p \cdot (1 - S))$ . It also expects that alpha.nonins has been
normalized (to sum to 1) ahead of time.

MBS.likelihood.partialrealloc.numerical.integration.nonins.noninsertonlypart.vac.forOneR
eallocatedMass <- function ( .reallocated.mass.given.nonins )#,
.optional.for.debugging.the.new.insprob )
{
stopifnot( abs( 1 - sum( alpha.nonins ) ) <= 1E-7 ); ## DEHACKIFY MAGIC #s
# This ends up being just the multinomial prob using a modified version of the
alphas.
# First make the probs all add up to 1-.reallocated.mass.given.nonins
.alpha.nonins <- alpha.nonins * ( 1 - .reallocated.mass.given.nonins );
# Now divide the remaining bit (which is .reallocated.mass) evenly among them..
.alpha.nonins <- .alpha.nonins + ( .reallocated.mass.given.nonins / length(
.alpha.nonins ) );
# Should add to 1 now.
stopifnot( abs( 1 - sum( .alpha.nonins ) ) <= 1E-7 ); ## DEHACKIFY MAGIC #s

return( dmultinom( vac.counts.nonins, prob = .alpha.nonins, log = FALSE ) );
}
MBS.likelihood.partialrealloc.numerical.integration.nonins.noninsertonlypart.vac.forOneR
eallocatedMass (..)

# Note .evaluate.atS.local should be of length 1!
MBS.likelihood.partialrealloc.numerical.integration.evaluate.at0 <- function (
evaluate.at0, .evaluate.atS.local )
{
# Note evaluate.at0 will be a vector (when called via integrate(..)).

```

```

rv <-
  dbinom( vac.counts.ins[ 1 ], sum( vac.counts.ins ), ( evaluate.at0 * ( 1 -
.evaluate.atS.local ) ), log = FALSE );

  return( rv );
} # MBS.likelihood.partialrealloc.numerical.integration.evaluate.at0 (..)

# Note .reallocated.mass should be of length 1!
ret <- sapply( evaluate.atS, function ( .evaluate.atS.local ) {
  .evaluate.at0.local <- alpha.ins[ 1 ] / sum( alpha.ins );
  rvS.ins <- MBS.likelihood.partialrealloc.numerical.integration.evaluate.at0(
.evaluate.at0.local, .evaluate.atS.local );
  ## Calculate the noninsertonly part of the nonins prob for vaccine recipients (which
does depend on S). See above where we've already calculated the insertonlypart, and
where we've pre-normalized alpha.nonins.
  ## Note that what we pass to the fn is the removed mass (S*p) as a fraction of the
noninsert mass (1-p*(1-S)).
  r0.nonins.noninsertonlypart.vac <-
MBS.likelihood.partialrealloc.numerical.integration.nonins.noninsertonlypart.vac.forOneR
eallocatedMass( ( .evaluate.at0.local * .evaluate.atS.local ) / ( 1 - ( .evaluate.at0.local * (
1-.evaluate.atS.local ) ) ) );
  rv.probOfS <-
  dbeta( .evaluate.atS.local, alphaS[ 1 ], alphaS[ 2 ], log = FALSE );
  rvS.noninsertonlypart <- rv.probOfS * rvS.ins * r0.nonins.noninsertonlypart.vac;

  rvS.insertonlypart <- rv.probOfS * rvS.ins * r0.nonins.insertonlypart;
  rvS <- ( ( rvS.noninsertonlypart * ( 1 - insertonlymodel.prob ) ) + ( rvS.insertonlypart
* insertonlymodel.prob ) );
  .evaluated.Ss.and.values <-<- rbind( .evaluated.Ss.and.values, c(
.evaluate.atS.local, rvS, rvS.insertonlypart, rvS.noninsertonlypart ) );

  return( rvS );
} );

return( ret );
} # MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS (..)
# NOTE: As a side effect, calling this modifies .evaluated.Ss.and.values:
r <- integrate( MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS, 0, 1 );
.integral <- r$value; # We'll use this below when calculating .s.mean.
rv <- .integral;
if( log.results ) {
  rv <- log( rv );
}

if( return.isinsertonly.posterior | return.S.posterior.mode | return.S.posterior.mean |
return.S.posterior.median ) {
  ##### NOTE: This is assuming that alphaS = (1,1)
  stopifnot( all( alphaS == c( 1, 1 ) ) );
  .table <- .evaluated.Ss.and.values;
  .table <- .table[ sort( .table[ , 1 ], index.return=T )$ix, ];
  .table <- cbind( .table, .table[ , 2 ] / sum( .table[ , 2 ] ) );
}

```



```

.table <- cbind( .table, cumsum( .table[ , 5 ] ) );
colnames( .table ) <- c( "S", "P(d|S=s)", "P(d|S=s, isinsertonly=T)", "P(d|S=s,
isinsertonly=F)", "P(S=s|d)", "P(S<=s,d)" );

return.list <- list( density = rv );
if( return.S.posterior.mode ) {
  current.max.index <- which.max( .table[ , 5 ] );
  if( current.max.index == 1 ) {
    optimize.low <- 0;
    optimize.high <- .table[ current.max.index + 1, 1 ];
  } else if( current.max.index == nrow( .table ) ) {
    optimize.low <- .table[ current.max.index - 1, 1 ];
    optimize.high <- 1;
  } else {
    optimize.low <- .table[ current.max.index - 1, 1 ];
    optimize.high <- .table[ current.max.index + 1, 1 ];
  }
  .evaluated.Ss.and.values.old <- .evaluated.Ss.and.values;
  .evaluated.Ss.and.values <- matrix( ncol=4, nrow=0 );
  optimize.result <- optimize(
MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS, lower=optimize.low,
upper=optimize.high, maximum = TRUE, tol = S.mode.tolerance );

  return.list$S.mode <- optimize.result$maximum;
  ## While we're at it, also get the return value at the mode.
  rv.at.S.mode <- optimize.result$objective;

  ## If we're near an edge, check -- maybe we should be on the edge.
  if( optimize.low == 0 ) {
    rv.at.S.zero <- MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS( 0 );
    if( rv.at.S.zero >= rv.at.S.mode ) {
      return.list$S.mode <- 0;
      rv.at.S.mode <- rv.at.S.zero;
    }
  }
  if( optimize.high == 1 ) {
    rv.at.S.one <- MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS( 1 );
    if( rv.at.S.one >= rv.at.S.mode ) {
      return.list$S.mode <- 1;
      rv.at.S.mode <- rv.at.S.one;
    }
  }

  if( log.results ) {
    rv.at.S.mode <- log( rv.at.S.mode );
  }
  return.list$density.at.S.mode <- rv.at.S.mode;

  ## Now use the new evaluations to update the table.
  ## Now use the new evaluations to update the table.

```

```

        .evaluated.Ss.and.values      <-      rbind(      .evaluated.Ss.and.values.old,
    .evaluated.Ss.and.values );
    .table <- .evaluated.Ss.and.values;
    .table <- .table[ sort( .table[ , 1 ], index.return=T )$ix, ];
    .table <- cbind( .table, .table[ , 2 ] / sum( .table[ , 2 ] ) );
    .table <- cbind( .table, cumsum( .table[ , 5 ] ) );
    colnames( .table ) <- c( "S", "P(d|S=s)", "P(d|S=s, isinsertonly=T)", "P(d|S=s,
isinsertonly=F)", "P(S=s|d)", "P(S<=s,d)" );
    }
    if( return.S.posterior.mean ) {
        return.list$S.mean <- integrate( function( .evaluate.atS ) { return( .evaluate.atS * (
MBS.likelihood.partialrealloc.numerical.integration.evaluate.atS( .evaluate.atS ) /
.integral ) ); }, 0, 1 )$value;
    }
    if( return.S.posterior.median ) {
        ## NOTE that this is a fairly crude estimate, and I'm not interpolating, I'm just
averaging the values on either side of the estimated 50% point. So eg if one is at .499
and another is at .6, I take the simple average rather than interpolate.  TODO:
interpolate.
        return.list$S.median <- mean( c( .table[ which.max( .table[ .table[ , 6 ] <= .5, 6 ] ), 1 ],
.table[ which.max( .table[ .table[ , 6 ] < .5, 6 ] ) + 1, 1 ] ) );
    }
    if( return.isinsertonly.posterior ) {
        .rv.insertonly <- trapz( .table[ , 1 ], .table[ , 3 ] );
        .rv.noninsertonly <- trapz( .table[ , 1 ], .table[ , 4 ] );
        return.list$isinsertonly.posterior.prob <- ( .rv.insertonly * insertonlymodel.prob ) / ( (
.rv.insertonly * insertonlymodel.prob ) + ( .rv.noninsertonly * ( 1 - insertonlymodel.prob ) )
);
    }

    return( return.list );
} else {
    return( rv );
}
} # MBS.likelihood.partialrealloc.numerical.integration (..)

```

##### Actually running the analysis.. #####

```

scenario.genes <- c();
scenario.insert.file.names <- c();
scenario.run.names <- c();
scenario.site.filter.sets <- c();
scenario.sieve.methods <- c();
scenario.id <- 1;
for( the.gene in genes ) {
    inserts.for.gene <- grep( the.gene, dir( insert.dir ), value = T );
}

```



```

    if( sieve.method == "SMMB" ) { # uses all obs seqs
      vaccine.file.name <- paste( "rv144/26Aug2011/obs/", the.gene,
".aa.obs.111e.vac.fasta", sep = "" );
    } else { # uses one seq per subject
      vaccine.file.name <- paste( "rv144/26Aug2011/mindist/", the.gene,
".aa.mindist.1.vac.fasta", sep = "" );
    }
  }

  if( sieve.method == "SMMB" ) { # uses all obs seqs
    placebo.file.name <- paste( "rv144/26Aug2011/obs/", the.gene,
".aa.obs.111e.pla.fasta", sep = "" );
  } else { # uses one seq per subject
    placebo.file.name <- paste( "rv144/26Aug2011/mindist/", the.gene,
".aa.mindist.1.pla.fasta", sep = "" );
  }

  ## Load the hxb2 map
  hxb2.map <- read.delim( hxb2.map.file.name, sep="|" );

  ## Set up the screens/filters/masks.
  # We always filter out the nonalignable sites.
  nonalignable.sites.filter.filename <- paste( "rv144/26Aug2011/mask/", the.gene,
".hypervariableColumns.csv", sep = "" );
  filters.include.site <- !as.logical( read.csv( nonalignable.sites.filter.filename )[ , 2 ] );
  # We might also filter more (see above)
  for( additional.site.filter.filename in additional.site.filter.files[[ site.filter.set ] ] ) {
    additional.filter.include.site <- !as.logical( read.csv( additional.site.filter.filename )[ , 2 ]
);
    filters.include.site <- filters.include.site & additional.filter.include.site;
  }

  # These minimum-variability filters are insert-specific (so we need to load the seq
envs):
  vaccine.seq.envir <- readFastaFile( vaccine.file.name, be.verbose=be.verbose );
  placebo.seq.envir <- readFastaFile( placebo.file.name, be.verbose=be.verbose );
  insert.seq.envir <- readFastaFile( insert.file.name, be.verbose=be.verbose );
  if( sieve.method == "SMMB" ) {
    # We need to create the maps of sequences -> subjects; the seq
    # names start with the subject id, separated by a vertical
    # bar...
    vaccine.subject.ids <- gsub( "^(^\\|+)$", "\\1", ls( env=vaccine.seq.envir ) );
    vaccine.sequence.sets.list <- sapply( unique( vaccine.subject.ids ), function(
.subject.id ) { ls( env=vaccine.seq.envir )[ vaccine.subject.ids == .subject.id ] } );
    placebo.subject.ids <- gsub( "^(^\\|+)$", "\\1", ls( env=placebo.seq.envir ) );
    placebo.sequence.sets.list <- sapply( unique( placebo.subject.ids ), function(
.subject.id ) { ls( env=placebo.seq.envir )[ placebo.subject.ids == .subject.id ] } );
  } else {
    vaccine.sequence.sets.list <- NULL;
    placebo.sequence.sets.list <- NULL;
  }
}

```

```

    minimum.variability.filter.include.site <- chooseSitesToScreen.minimumVariability(
vaccine.seq.envir, placebo.seq.envir, insert.seq.envir,
vaccine.sequence.sets.list=vaccine.sequence.sets.list,
placebo.sequence.sets.list=placebo.sequence.sets.list );
# That's all we'll need those seq.envirs for. We reopen them in the below, but because
of refinement we probably need to reopen them anyway, so oh well.
rm( vaccine.seq.envir, placebo.seq.envir, insert.seq.envir );

include.site <- ( filters.include.site & minimum.variability.filter.include.site );

return.list <- list( scenario.id = scenario.id, seed = seed );

set.seed( seed );
if( sieve.method == "GWJ" ) {
  return.list$GWJ.result <- runGWJ( vaccine.file.name, placebo.file.name,
insert.file.name, gwj.weight.matrix.filename,
weight.matrix.is.probability.matrix=gwj.weight.matrix.is.probability.matrix,
include.site=include.site, hxb2.map=hxb2.map, run.name=run.name,
be.verbose=be.verbose, ... );
} else if( sieve.method == "SMMB" ) {
  return.list$SMMB.result <- runSMMB( vaccine.file.name, placebo.file.name,
insert.file.name, include.site=include.site, hxb2.map=hxb2.map, run.name=run.name,
be.verbose=be.verbose, vaccine.sequence.sets.list=vaccine.sequence.sets.list,
placebo.sequence.sets.list=placebo.sequence.sets.list, ... );
} else if( sieve.method == "MBS" ) {
  return.list$MBS.result <- runMBS( vaccine.file.name, placebo.file.name,
insert.file.name, include.site=include.site, hxb2.map=hxb2.map, run.name=run.name,
be.verbose=be.verbose, ... );
} else {
  stop( paste( "Unrecognized sieve.method:", sieve.method ) );
}

return( return.list );
} # run.scenario (..)

if( run.when.sourced ) {
  library( "foreach" )
  library( "doMC" )
  be.verbose <- FALSE;
  registerDoMC( multicore:::detectCores() - as.numeric( be.verbose ) ) # use all available
cores except maybe one (for i/o).
  scenarios.results <- foreach( scenario.id = 1:num.scenarios ) %dopar% { run.scenario(
scenario.id=scenario.id, be.verbose = be.verbose, seed = 98103 ) }
  names( scenarios.results ) <- scenario.run.names;
}

```