# TESTING FOR STOCHASTIC ORDERING ASSUMING MARGINAL COMPATIBILITY

ANIKO SZABO

## 1. Preliminaries

First we need to set up the C file so that it can access the R internals.

"..\src\ReprodCalcs.c" ?≡

```
        #include <stdlib.h>
        #include <R.h>
        #include <Rdefines.h>
        #include <Rmath.h>
        #include <R_ext/Applic.h>
        ◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

We will be using object of `CBData` class, which is defined in `CBData.w`.

## 2. Marginal compatibility

2.1. **Estimation.** The following C code implements the EM-algorithm for estimating the probabilities of response assuming marginal compatibility.

$$\pi_{rM}^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N} h(r_i, r, n_i)\frac{\pi_{r,M}^{(t)}}{\pi_{r_i,n_i}^{(t)}}, \tag{1}$$

First we write a help-function that calculates all the probabilities $\pi_{r,n}$ given the set of $\theta_r = \pi_{r,M}$. While there are a variety of ways doing this, we use a recursive formula:

$$\pi_{r,n} = \frac{n+1-r}{n+1}\pi_{r,n+1} + \frac{r+1}{n+1}\pi_{r+1,n+1} \tag{2}$$

"..\src\ReprodCalcs.c" ?≡

```
        double static **Marginals(double* theta, int maxsize)
        {
         double **res;
         int r, n;

         res = malloc((maxsize+1)*sizeof(double *));
         for (n=0; n<=maxsize; n++){
                res[n] = calloc(n+1, sizeof(double));
         }

         for (r=0; r<=maxsize; r++){
                res[maxsize][r] = theta[r];
         }

         for (n=maxsize-1; n>=1; n--){
                for (r=0; r<=n; r++){
                        res[n][r] = (n+1.0-r)/(n+1.0)*res[n+1][r] + (r+1.0)/(n+1.0)*res[n+1][r+1];
```

*Date*: September 28, 2023.

```
         }
 }

 return res;
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

The actual EM iterations are performed in `ReprodEstimates`.

"..\src\ReprodCalcs.c" ?≡

```
SEXP ReprodEstimates(SEXP nvec, SEXP rvec, SEXP freqvec)
{
 double *theta, *thetanew, abserror, **marg;
 int i, maxsize, nr, ntot, r, ri, ni, n, fri, start;
 SEXP res;
 const double eps=1e-16;

 nr = LENGTH(nvec);
 maxsize = 0;
 ntot = 0;
 for (i=0; i<nr; i++){
         if (INTEGER(nvec)[i]>maxsize){
                 maxsize = INTEGER(nvec)[i];
         }
         ntot += INTEGER(freqvec)[i];
 }

 theta = malloc((maxsize+1) * sizeof(double));
 thetanew = malloc((maxsize+1) * sizeof(double));
 //starting values
 for (r=0; r<=maxsize; r++){
         theta[r] = 1.0/(maxsize+1);
 }
 abserror = 1;
 //EM update
 while (abserror>eps){
         abserror = 0;
         marg = Marginals(theta, maxsize);
         for (r=0; r<=maxsize; r++) thetanew[r] = 0;
         for (i=0; i<nr; i++){
                 ri = INTEGER(rvec)[i];
                 ni = INTEGER(nvec)[i];
                 fri = INTEGER(freqvec)[i];
                 for (r=ri; r<=maxsize-ni+ri; r++){
                         thetanew[r] += choose(ni,ri)*choose(maxsize-ni,r-ri)*theta[r]*fri*1.0/
                                        marg[ni][ri] ;
                 }
         }
         for (r=0; r<=maxsize; r++){
                 thetanew[r] = thetanew[r]/(ntot*choose(maxsize,r)*1.0);
                 abserror += fabs(thetanew[r]-theta[r]);
                 theta[r] = thetanew[r];
         }
         for(n = 0; n <= maxsize; n++) free(marg[n]);
     free(marg);
 }
 PROTECT(res = allocMatrix(REALSXP, maxsize+1, maxsize));
 marg = Marginals(theta, maxsize);
 for (n=1, start=0; n<=maxsize; n++, start+=(maxsize+1)){
         for (r=0; r<=n; r++){
                 REAL(res)[start+r] = marg[n][r];
         }
         for (r=n+1; r<=maxsize; r++){
                 REAL(res)[start+r] = NA_REAL;
         }
 }

 for(n = 0; n <= maxsize; n++) free(marg[n]);
 free(marg);
 free(theta);
 free(thetanew);

 UNPROTECT(1);
 return res;

}
```
◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `ReprodEstimates` ?.

We will call the C function from R to calculate estimates separately by treatment group. For the package the compiled library needs to be loaded.

"../R/aaa-generics1.R" ?≡

```
#'Distribution of the number of responses assuming marginal compatibility.
#'
#'The \code{mc.est} function estimates the distribution of the number of
#'responses in a cluster under the assumption of marginal compatibility:
#'information from all cluster sizes is pooled. The estimation is performed
#'independently for each treatment group.
#'
#'The EM algorithm given by Stefanescu and Turnbull (2003) is used for the binary data.
#'
#'@useDynLib CorrBin, .registration=TRUE
#'@export
#'@param object a \code{\link{CBData}} or \code{\link{CMData}} object
#'@param \dots other potential arguments; not currently used
#'@return For \code{CBData}: A data frame giving the estimated pdf for each treatment and
#'clustersize.  The probabilities add up to 1
#'for each \code{Trt}/\code{ClusterSize} combination. It has the following columns:
#'@return \item{Prob}{numeric, the probability of \code{NResp} responses in a
#'cluster of size \code{ClusterSize} in group \code{Trt}}
#'@return \item{Trt}{factor, the treatment group}
#'@return \item{ClusterSize}{numeric, the cluster size}
#'@return \item{NResp}{numeric, the number of responses}
#'@author Aniko Szabo
#'@references Stefanescu, C. & Turnbull, B. W. (2003) Likelihood inference for
#'exchangeable binary data with varying cluster sizes.  \emph{Biometrics}, 59,
#'18-24
#'@keywords nonparametric models
#'@examples
#'
#'data(shelltox)
#'sh.mc <- mc.est(shelltox)
#'
#'library(lattice)
#'xyplot(Prob~NResp|factor(ClusterSize), groups=Trt, data=sh.mc, subset=ClusterSize>0,
#'    type="l", as.table=TRUE, auto.key=list(columns=4, lines=TRUE, points=FALSE),
#'    xlab="Number of responses", ylab="Probability P(R=r|N=n)")
#'
#'@name mc.est

mc.est <- function(object,...) UseMethod("mc.est")
```

◇

File defined by ?, ?.
Uses: mc.est ?.

"../R/Reprod.R" ?≡

```
#'@rdname mc.est
#'@method mc.est CBData
#'@export

mc.est.CBData <- function(object, ...){
  cbdata <- object[object$Freq>0, ]
  #by trt
  do.est.fun <- function(x){
    est <- .Call("ReprodEstimates", as.integer(x$ClusterSize), as.integer(x$NResp),
                         as.integer(x$Freq),PACKAGE="CorrBin")
    est <- cbind(c(1,rep(NA,nrow(est)-1)), est)
    idx <- upper.tri(est,diag=TRUE)
    est.d <- data.frame(Prob=est[idx], ClusterSize=as.integer(col(est)[idx]-1),
                        NResp=as.integer(row(est)[idx]-1),
                        Trt=x$Trt[1])
    est.d}

  est.list <- by(cbdata, list(Trt=cbdata$Trt), do.est.fun)
  do.call(rbind, est.list)}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `mc.est` ?, ?, ?, ?, ?, ?.
Uses: `ReprodEstimates` ?.

2.2. **Testing marginal compatibility.** The `mc.test.chisq` function implements Pang and Kuk's version of the test for marginal compatibility. Note that it only tests that the marginal probability of response $p_i$ does not depend on the cluster size. The original test was only defined for one group and the test statistic was compared to $\chi^2_1$ (or more precisely, it was a z-test), however the test is easily generalized by adding the test statistics for the $G$ separate groups and using a $\chi^2_G$ distribution.

$$Z_g = \Big[\sum_{i=1}^{N_g}(c_{n_{g,i}} - \bar{c}_g)r_{g,i}\Big] \Big/ \Big[\hat{p}_g(1-\hat{p}_g)\sum_{i=1}^{N_g} n_{g,i}(c_{n_{g,i}} - \bar{c}_g)^2\{1+(n_{g,i}-1)\hat{\rho}_g\}\Big]^{1/2}, \tag{3}$$

where $c_n$ are the scores for the Cochran-Armitage test usually chosen as $c_n = n - (M+1)/2$, $\bar{c}_g = \big(\sum_{i=1}^{N_g} n_{g,i}c_{n_{g,i}}\big)/\big(\sum_{i=1}^{N_g} n_{g,i}\big)$ is a weighted average of the scores; $\hat{p}_g = \big(\sum_{i=1}^{N_g} r_{g,i}\big)/\big(\sum_{i=1}^{N_g} n_{g,i}\big)$ is the raw response probability, and $\hat{\rho}_g = 1 - \big[\sum_{i=1}^{N_g}(n_{g,i}-r_{g,i})r_{g,i}/n_{g,i}\big]/\big[\hat{p}_g(1-\hat{p}_g)\sum_{i=1}^{N_g}(n_{g,i}-1)\big]$ is the Fleiss-Cuzack estimate of the intra-cluster correlation for the $g$th treatment group.

$$X^2 = \sum_{g=1}^{G} Z_g^2 \sim \chi^2_G \text{ under } H_0. \tag{4}$$

"../R/aaa-generics1.R" ?≡

```
#’Test the assumption of marginal compatibility
#’
#’\code{mc.test.chisq} tests whether the assumption of marginal compatibility is
#’violated in the data.
#’
#’The assumption of marginal compatibility (AKA reproducibility or interpretability) implies that
#’the marginal probability of response does not depend on clustersize.
#’Stefanescu and Turnbull (2003), and Pang and Kuk (2007) developed a
#’Cochran-Armitage type test for trend in the marginal probability of success
#’as a function of the clustersize. \code{mc.test.chisq} implements a
#’generalization of that test extending it to multiple treatment groups.
#’
#’@export
#’@param object a \code{\link{CBData}} or \code{\link{CMData}} object
#’@param \dots other potential arguments; not currently used
#’@return A list with the following components:
#’@return \item{overall.chi}{the test statistic; sum of the statistics for each
#’group}
#’@return \item{overall.p}{p-value of the test}
#’@return \item{individual}{a list of the results of the test applied to each
#’group separately: \itemize{ \item chi.sq the test statistic for the group
#’\item p p-value for the group}}
#’@author Aniko Szabo
```

```
#'@seealso \code{\link{mc.est}} for estimating the distribution under marginal
#'compatibility.
#'@references Stefanescu, C. & Turnbull, B. W. (2003) Likelihood inference for
#'exchangeable binary data with varying cluster sizes. \emph{Biometrics}, 59,
#'18-24
#'
#'Pang, Z. & Kuk, A. (2007) Test of marginal compatibility and smoothing
#'methods for exchangeable binary data with unequal cluster sizes.
#'\emph{Biometrics}, 63, 218-227
#'@keywords htest
#'@examples
#'
#'data(shelltox)
#'mc.test.chisq(shelltox)
#'

mc.test.chisq <- function(object,...) UseMethod("mc.test.chisq")
```

◇

File defined by ?, ?.
Uses: `mc.est` ?, `mc.test.chisq` ?.

`"../R/Reprod.R"` ?≡

```
#'@rdname mc.test.chisq
#'@method mc.test.chisq CBData
#'@export
#'@importFrom stats pchisq

mc.test.chisq.CBData <- function(object,...){
  cbdata <- object[object$Freq>0, ]

  get.T <- function(x){
      max.size <- max(x$ClusterSize)
      scores <- (1:max.size) - (max.size+1)/2
      p.hat <- with(x, sum(Freq*NResp) / sum(Freq*ClusterSize))
      rho.hat <- with(x, 1-sum(Freq*(ClusterSize-NResp)*NResp/ClusterSize) /
          (sum(Freq*(ClusterSize-1))*p.hat*(1-p.hat)))  #Fleiss-Cuzick estimate
      c.bar <- with(x, sum(Freq*scores[ClusterSize]*ClusterSize) / sum(Freq*ClusterSize))
      T.center <- with(x, sum(Freq*(scores[ClusterSize]-c.bar)*NResp))
      Var.T.stat <-  with(x,
         p.hat*(1-p.hat)*sum(Freq*(scores[ClusterSize]-c.bar)^2*ClusterSize*(1+(ClusterSize-1)*rho.hat)))
      X.stat <- (T.center)^2/Var.T.stat
      X.stat}

   chis <- by(cbdata, cbdata$Trt, get.T)
   chis <- chis[1:length(chis)]
   chi.list <- list(chi.sq=chis, p=pchisq(chis, df=1, lower.tail=FALSE))
   overall.chi <- sum(chis)
   overall.df <- length(chis)
   list(overall.chi=overall.chi, overall.p=pchisq(overall.chi, df=overall.df, lower.tail=FALSE),
        individual=chi.list)
}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `mc.test.chisq` ?.

## 3. Estimation under stochastic ordering

We implement the estimation of the stochastically ordered MLE based on the mixture representation following Hoff. The parameter `turn` controls the extension to unimodal down-then-up orderings. The default value of `turn=1` corresponds to the stochastically ordered $H_a$.

The actual calculations will be done in C; the following function sets up the call from R. The input dataset should have variables `NResp`, `ClusterSize`,`Trt` and `Freq` – it is most easily achieved by creating a "CBData" object.

The computational details are controlled by setting the `control` argument. It should be a list with parameter settings; the simplest way to generate a correct list is a call to the `soControl()` function. `S` gives the basis of the mixing distribution – its rows are all the possible non-decreasing vectors (see below for how it is obtained). `Q` is the mixing distribution, it is a $G$-dimensional matrix ($G$ is the number of treatment groups). For the ISDM method it is initialized so that the constant vectors of $S$ are given equal probability ($H_0$ puts weight only on the constant vectors), that is $Q[0,0,\ldots,0] = Q[1,1,\ldots,1] = \cdots = Q[N,N,\ldots,N] = 1/(N+1)$. For the EM method we have to put some weight on all possible $\mathbf{v}$ vectors, because it changes $Q$ only multiplicatively.

"../R/Reprod.R" ?≡

```
#'Order-restricted MLE assuming marginal compatibility
#'
#'\code{SO.mc.est} computes the nonparametric maximum likelihood estimate of
#'the distribution of the number of responses in a cluster \eqn{P(R=r|n)} under
#'a stochastic ordering constraint. Umbrella ordering can be specified using
#'the \code{turn} parameter.
#'
#'Two different algorithms: EM and ISDM are implemented. In general, ISDM (the
#'default) should be faster, though its performance depends on the tuning
#'parameter \code{max.directions}: values that are too low or too high slow the
#'algorithm down.
#'
#'\code{SO.mc.est} allows extension to an umbrella ordering: \eqn{D_1 \geq^{st}
#'\cdots \geq^{st} D_k \leq^{st} \cdots \leq^{st} D_n}{D_1 >= \ldots >= D_k <=
#'\ldots <= D_n} by specifying the value of \eqn{k} as the \code{turn}
#'parameter. This is an experimental feature, and at this point none of the
#'other functions can handle umbrella orderings.
#'
#'@useDynLib CorrBin, .registration=TRUE
#'@export
#'@importFrom stats xtabs
#'@param cbdata an object of class \code{\link{CBData}}.
#'@param turn integer specifying the peak of the umbrella ordering (see
#'Details). The default corresponds to a non-decreasing order.
#'@param control an optional list of control settings, usually a call to
#'\code{\link{soControl}}.  See there for the names of the settable control
#'values and their effect.
#'@return A list with components:
#'
#'Components \code{Q} and \code{D} are unlikely to be needed by the user.
#'@return \item{MLest}{data frame with the maximum likelihood estimates of
#'\eqn{P(R_i=r|n)}}
#'@return \item{Q}{numeric matrix; estimated weights for the mixing distribution}
#'@return \item{D}{numeric matrix; directional derivative of the log-likelihood}
#'@return \item{loglik}{the achieved value of the log-likelihood}
#'@return \item{converge}{a 2-element vector with the achieved relative error and
#'the performed number of iterations}
#'@author Aniko Szabo, aszabo@@mcw.edu
#'@seealso \code{\link{soControl}}
#'@references Szabo A, George EO. (2010) On the Use of Stochastic Ordering to
```

```
#'Test for Trend with Clustered Binary Data. \emph{Biometrika} 97(1), 95-108.
#'@keywords nonparametric models
#'@examples
#'
#'  data(shelltox)
#'  ml <- SO.mc.est(shelltox, control=soControl(eps=0.01, method="ISDM"))
#'  attr(ml, "converge")
#'
#'  require(lattice)
#'  panel.cumsum <- function(x,y,...){
#'    x.ord <- order(x)
#'    panel.xyplot(x[x.ord], cumsum(y[x.ord]), ...)}
#'
#'  xyplot(Prob~NResp|factor(ClusterSize), groups=Trt, data=ml, type="s",
#'       panel=panel.superpose, panel.groups=panel.cumsum,
#'       as.table=TRUE, auto.key=list(columns=4, lines=TRUE, points=FALSE),
#'       xlab="Number of responses", ylab="Cumulative Probability R(R>=r|N=n)",
#'       ylim=c(0,1.1), main="Stochastically ordered estimates\n with marginal compatibility")
#'
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: mc.est ?, SO.mc.est ?, soControl ?.

"../R/Reprod.R" ?≡

```
SO.mc.est <- function(cbdata, turn=1, control=soControl()){
  tab <- xtabs(Freq~factor(ClusterSize,levels=1:max(ClusterSize))+
                factor(NResp,levels=0:max(ClusterSize))+Trt, data=cbdata)
        size <- dim(tab)[1]
        ntrt <- dim(tab)[3]
        ntot <- sum(tab)
  storage.mode(tab) <- "double"
        Q <- array(0, dim=rep(size+1,ntrt))
        storage.mode(Q) <- "double"

        S <- DownUpMatrix(size, ntrt, turn)
        storage.mode(S) <- "integer"

        if ((control$start=="H0")&(control$method=="EM")){
          warning("The EM algorithm can only use 'start=uniform'. Switching options.")
          start <- "uniform"
  }
        if (control$start=="H0"){
          const.row <- matrix(0:size, nrow=size+1, ncol=ntrt)
          Q[const.row+1] <- 1/(size+1)
                }
        else {  #start=="uniform"
          Q[S+1] <- 1/(nrow(S))
    }

  res0 <- switch(control$method,
      EM = .Call("MixReprodQ", Q, S, tab, as.integer(control$max.iter), as.double(control$eps),
                                   as.integer(control$verbose), PACKAGE="CorrBin"),
      ISDM = .Call("ReprodISDM", Q, S, tab, as.integer(control$max.iter), as.integer(control$max.directio
                   as.double(control$eps),  as.integer(control$verbose), PACKAGE="CorrBin"))

  names(res0) <- c("MLest","Q","D","loglik", "converge")
  names(res0$converge) <- c("rel.error", "n.iter")
  res <- res0$MLest

  dimnames(res) <- list(NResp=0:size, ClusterSize=1:size, Trt=1:ntrt)
  res <- as.data.frame.table(res)
  names(res) <- c("NResp","ClusterSize","Trt","Prob")
  res$NResp  <- as.numeric(as.character(res$NResp))
  res$ClusterSize  <- as.numeric(as.character(res$ClusterSize))
  res <- res[res$NResp <= res$ClusterSize,]
  levels(res$Trt) <- levels(cbdata$Trt)

  attr(res, "loglik") <- res0$loglik
  attr(res, "converge") <- res0$converge
  res
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: SO.mc.est ?, ?.
Uses: DownUpMatrix ?, mc.est ?, ReprodISDM ?, soControl ?.

The values supplied in the call to soControl replace the defaults and a list with all possible arguments is returned. The returned list is used as the control argument to the SO.mc.est function.

The method argument allows to select either the EM, or the ISDM method.

"../R/Reprod.R" ?≡

```
#'Control values for order-constrained fit
#'
#'The values supplied in the function call replace the defaults and a list with
#'all possible arguments is returned.  The returned list is used as the control
#'argument to the \code{\link{mc.est}}, \code{\link{SO.LRT}}, and
#'\code{\link{SO.trend.test}} functions.
#'
#'@export
#'@param method a string specifying the maximization method
#'@param eps a numeric value giving the maximum absolute error in the
#'log-likelihood
#'@param max.iter an integer specifying the maximal number of iterations
#'@param max.directions an integer giving the maximal number of directions
#'considered at one step of the ISDM method.  If zero or negative, it is set to
#'the number of non-empty cells. A value of 1 corresponds to the VDM algorithm.
#'@param start a string specifying the starting setup of the mixing
#'distribution; "H0" puts weight only on constant vectors (corresponding to the
#'null hypothesis of no change), "uniform" puts equal weight on all elements.
#'Only a "uniform" start can be used for the "EM" algorithm.
#'@param verbose a logical value; if TRUE details of the optimization are
#'shown.
#'@return a list with components for each of the possible arguments.
#'@author Aniko Szabo aszabo@@mcw.edu
#'@seealso \code{\link{mc.est}}, \code{\link{SO.LRT}},
#'\code{\link{SO.trend.test}}
#'@keywords models
#'@examples
#'
#'# decrease the maximum number iterations and
#'# request the "EM" algorithm
#' soControl(method="EM", max.iter=100)
#'
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: `mc.est` ?, `SO.trend.test` ?, `soControl` ?.

`"../R/Reprod.R"` ?≡

```
soControl <- function(method=c("ISDM","EM"), eps=0.005, max.iter=5000,
      max.directions=0, start=ifelse(method=="ISDM", "H0", "uniform"), verbose=FALSE){
  method <- match.arg(method)
  start <- match.arg(start, c("uniform","H0"))
  list(method = match.arg(method), eps = eps, max.iter = max.iter,
       max.directions = max.directions, start=start, verbose = verbose)
}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `soControl` ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

The `makeSmatrix` function creates a matrix the rows of which are all the possible non-decreasing vectors. The idea is the following: there is a one-to-one correspondence between a non-decreasing sequence of length $k$ using the values $0, 1, \ldots, n$ and combinations of $k$ elements out of $n + k$. Imagine $n$ balls and $k$ sticks; each of the possible arrangements of theses $n + k$ objects into a row is a combination. We can transform it to a nondecreasing sequence by counting the number of balls to the left of the first stick, second stick, etc. For example, with $n = 5$ and $k = 4$, a possible arrangement is $\circ | \circ || \circ \circ \circ |$; this corresponds to the sequence (1,2,2,5). The `Comb` function generates all possible combinations of $k$ elements out of $N = n + k$ elements by recursion on $N$, and when a combination is ready, it transforms it to a non-decreasing sequence and puts it into the output matrix. The code for generating the combinations was written by Joe Sawada, 1997 and obtained from the Combinatorial Object Server (http://theory.cs.uvic.ca/inf/comb/CombinationsInfo.html), but was rewritten without nested functions.

`"..\src\ReprodCalcs.c"` ?≡

```
void Comb(int j, int m, int nn, int kk, int nS, int* a, int* pos, SEXP res) {
        int i, val, step;
        if (j > nn) {
        ⟨ Convert 'a' to non-decreasing sequence and insert into 'res' ? ⟩
    }
        else {
                if (kk-m < nn-j+1) {
                        a[j] = 0; Comb(j+1, m, nn, kk, nS, a, pos, res);
                }
                if (m<kk) {
                        a[j] = 1; Comb(j+1, m+1, nn, kk, nS, a, pos, res);
                }
        }
}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `Comb` ?.

⟨ *Convert 'a' to non-decreasing sequence and insert into 'res'* ? ⟩ ≡

```
                          val = 0;
                          step = 0;
                          for (i=1; i<=nn; i++) {
                                  if (a[i]==1){
                                          INTEGER(res)[*pos+step]=val;
                                          step += nS;
                                  }
                                  else { //a[i]=0;
                                          val++;
                                  }
                          }
                      *pos = *pos+1;
```

◇
Fragment referenced in ?.

"..\src\ReprodCalcs.c" ?≡

```
SEXP makeSmatrix(SEXP size, SEXP ntrt){
int *a, i, pos, nn, kk, nS;
SEXP res;
   nn = asInteger(size) + asInteger(ntrt);
   kk = asInteger(ntrt);
   a = calloc(nn+1, sizeof(int));
   for (i=1; i<=kk; i++) a[i]=1;

   nS = choose(nn,kk);
   PROTECT(res = allocMatrix(INTSXP, nS, kk));
   pos = 0;

   Comb(1, 0, nn, kk, nS, a, &pos, res);

   UNPROTECT(1);
   free(a);

  return res;
}
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `makeSmatrix` ?, ?, ?.
Uses: `Comb` ?.

`DownUpMatrix` extends the capabilities of `makeSmatrix` to generate vectors that have $q$ non-increasing elements followed $k - q$ non-decreasing elements.

The general outline of the algorithm is as follows ($n$ is `size`, $k$ is `ntrt`, and $q$ is `turn`):

"../R/Reprod.R" ?≡

```
#'Internal CorrBin objects
#'
#'Internal CorrBin objects.
#'
#'These are not to be called by the user.
#'
#'@rdname CorrBin-internal
#'@aliases .required DownUpMatrix
#'@keywords internal
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: `DownUpMatrix` ?.

"../R/Reprod.R" ?≡

```
DownUpMatrix <- function(size, ntrt, turn){
  if ((turn<1)|(turn>ntrt)) stop("turn should be between 1 and ntrt")
  ⟨ Take care of turn=1 and turn=ntrt ? ⟩
  ⟨ Generate non-increasing sequences of length turn with values ≤ size ? ⟩
  res2list <- list()
  for (sq in 0:size){
    ⟨ Generate non-decreasing sequences of length ntrt-turn with values between sq and size ? ⟩
  }
  ⟨ Combine the two parts of the sequences ? ⟩
}
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `DownUpMatrix` ?, ?.

Non-increasing sequences are generated by subtracting non-decreasing sequences from $n$:
⟨ *Generate non-increasing sequences of length turn with values* $\leq$ *size* ? ⟩ ≡

```
  res1 <- .Call("makeSmatrix", as.integer(size), as.integer(turn),PACKAGE="CorrBin")
  res1 <- size - res1;
```

◇
Fragment referenced in ?.
Uses: `makeSmatrix` ?.

`res2list` will be a list of matrices defining non-decreasing sequences for all possible starting points $s_q$. We get them by generating all non- decreasing sequences with values $\leq n - s_q$ and then adding $s_q$.
⟨ *Generate non-decreasing sequences of length ntrt-turn with values between sq and size* ? ⟩ ≡

```
  S <- .Call("makeSmatrix", as.integer(size-sq), as.integer(ntrt-turn),PACKAGE="CorrBin")
  res2list <- c(res2list, list(sq+S))
```

◇
Fragment referenced in ?.
Uses: `makeSmatrix` ?.

To match all possible starts with all possible ends, we split `res1` into a list based on the last value ($s_q$) and do a Cartesian product (everything-to- everything merge) with the corresponding elements of `res2list`:
⟨ *Combine the two parts of the sequences* ? ⟩ ≡

```
  res1list <- by(res1, res1[,turn], function(x)x)
  res <- mapply(merge, res1list, res2list, MoreArgs=list(by=NULL), SIMPLIFY=FALSE)
  res <- data.matrix(do.call(rbind, res))
  rownames(res) <- NULL
  colnames(res) <- NULL
  res
```

◇
Fragment referenced in ?.

When $q = 1$ or $q = k$, no merging is needed and we can speed up the algorithm by treating them as special cases:

⟨ *Take care of turn=1 and turn=ntrt ?* ⟩ ≡

```
if (turn==1){
   res <- .Call("makeSmatrix", as.integer(size), as.integer(ntrt),PACKAGE="CorrBin")
   return(res)
}
if (turn==ntrt){
   res <- .Call("makeSmatrix", as.integer(size), as.integer(ntrt),PACKAGE="CorrBin")
   return(size - res)
}
```

◇

Fragment referenced in ?.
Uses: `makeSmatrix` ?.

### 3.1. **ISDM estimation.**

In the ISDM algorithm, at each step we move toward a new mixing distribution by maximizing

$$l\big(\alpha_0 Q^{(t)} + \sum_i \alpha_i \delta_{\mathbf{v}_t^i}\big), \tag{5}$$

where $\mathbf{v}_t^1, \ldots, \mathbf{v}_t^m$ corresponding to the at most $m_0$ largest positive values of the directional derivative $D_{Q^{(t)}}(\mathbf{v})$, where

$$D_Q(\mathbf{v}) = \sum_{g,i} \frac{h(r_{g,i}, v_g, n_{g,i})}{\sum_{\mathbf{q}} h(r_{g,i}, q_g, n_{g,i}) Q(\mathbf{q})} - N. \tag{6}$$

First we define a variety of help-functions.

`GetTabElem` allows easy access using three indices to the 3-dimensional data matrix `tab` that gets converted to a long vector when it is passed from R. Another option would have been to set up a C-style array of pointers to pointers to rows.

`"..\src\ReprodCalcs.c"` ?≡

```
double GetTabElem(SEXP tab, int size, int n, int r, int j){
         return REAL(tab)[(n-1)+size*(r+(size+1)*j)];
 }
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

`HyperTable` makes a 3-dimensional C-style array with the hyper-geometric probabilities $h(i,j,k) = \binom{j}{i}\binom{N-j}{k-i}/\binom{N}{k}$. R's `dhyper` function does the actual calculations.

`"..\src\ReprodCalcs.c"` ?≡

```
double ***HyperTable(int size){
// dhyper(i, j, size-j, k), i=0:size; j=0:size; k=0:size
 double ***res;
 int i, j, k;

 res = malloc((size+1)*sizeof(double*));
 for (i=0; i<=size; i++){
         res[i] = malloc((size+1)*sizeof(double*));
         for (j=0; j<=size; j++) res[i][j] = calloc(size+1, sizeof(double));
 }

 for (i=0; i<=size; i++){
         for (j=i; j<=size; j++){
                 for (k=i; k<=size-j+i; k++)
                         res[i][j][k] = dhyper(i, j, size-j, k, 0);
                 }
         }
 return res;
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `HyperTable` ?, ?, ?.

The matrix $S$ created by `makeSmatrix` contains the numeric values of the non-decreasing vectors. `IndexVector` and `IndexVectorC` create a vector of indices corresponding to the location of each row of $S$ in a vectorized G-dimensional $N \times N \times \cdots \times N$ array $(Q)$. That is if a row of $S$ is $(0, 1, 4)$, then the corresponding element of the index vector will point to the [0,1,4]th element of a $N \times N \times N$ matrix that is represented by a long (length= $N^3$) vector. This setup is needed as the number of dimensions of $Q$ depends on the number of treatments, so it is not known at compile time.

The difference between the two functions is that `IndexVector` works for an R-style matrix (that has been converted to a vector during the transfer), while `IndexVectorC` works for a C-style matrix (an array of pointers to arrays).

`"..\src\ReprodCalcs.c"` ?≡

```
int static *IndexVectorC(int **S, int N, int G, int nrowS){
 int *idx, i, j;

 idx = calloc(nrowS, sizeof(int));

 for (j=G-1; j>=0; j--){
   for (i=0; i<nrowS; i++){
           idx[i] = N*idx[i] + S[i][j];
   }
 }

 return idx;
}
int static *IndexVector(SEXP S, int N, int G, int nrowS){
 int *idx, i, j,  start;

 idx = calloc(nrowS, sizeof(int));

 for (j=G-1, start=(G-1)* nrowS; j>=0; j--, start -= nrowS){
   for (i=0; i<nrowS; i++){
           idx[i] = N*idx[i] + INTEGER(S)[start+i];
   }
 }

 return idx;
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `IndexVector`, Never used, `IndexVectorC` ?.

`CalcMarginals` calculates the marginal probabilities in the denominator of (13)

$$Q_{gn}(r) = P(r \text{ responses } | \text{treatment } g, \text{cluster size } n) = \sum_{\mathbf{q}} h(r, q_g, n) Q^{(t)}(\mathbf{q}) \tag{7}$$

for $j = 1, \ldots, G$, $n = 1, \ldots, N$, $r = 0, \ldots, n$. It returns a 3-dimensional C-style matrix.

`"..\src\ReprodCalcs.c"` ?≡

```
double ***CalcMarginals(SEXP S, SEXP Q, double ***ht, int *idx, int ntrt, int size, int nS){
   int j, i, n, x, start, sj;
         double ***marg;
         //ht is from HyperTable, it is passed to avoid recalculation

         marg = malloc(ntrt*sizeof(double*));
         for (j=0; j<ntrt; j++){
                 marg[j] = malloc((size+1)*sizeof(double*));
                 for (n=1; n<=size; n++) marg[j][n] = calloc(n+1, sizeof(double));
         }

         for (i=0; i<nS; i++){
                 for (j=0, start=0; j<ntrt; j++, start+=nS){
                         sj = INTEGER(S)[start+i];
                         for (n=1; n<=size; n++){
                                 for (x=imax2(0,sj-size+n); x<=imin2(sj,n); x++){
                                         marg[j][n][x] += REAL(Q)[idx[i]]*ht[x][n][sj];
                                 }
                         }
                 }
         }
  return marg;
 }
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `CalcMarginals` ?, ?, ?.
Uses: `HyperTable` ?.

`CalcD` calculates the directional derivative of the log-likelihood into the direction of each possible **v** (that is for each row of $S$) as defined in (6). Note that the denominator contains the marginal probabilities defined above. The function does not have a return value, the first parameter $D$ is modified (or rather, the value it points to is modified).

`"..\src\ReprodCalcs.c"` ?≡

```
void CalcD(SEXP D, SEXP S, SEXP tab, int *idx, double ***ht, double ***marg, int ntrt,
           int nS, int size, int ntot){
   int j, i, n, x, sj, start, t;
         //ntot = sum(tab) -- passing it avoids having to recalculate it
         for (i=0; i<nS; i++){
                 REAL(D)[idx[i]] = -ntot;
                 for (j=0, start=0; j<ntrt; j++, start+=nS){
                         sj = INTEGER(S)[start+i];
                         for (n=1; n<=size; n++){
                                 for (x=imax2(0,sj-size+n); x<=imin2(sj,n); x++){
                                         t = GetTabElem(tab,size,n,x,j);
                                         if (t>0) REAL(D)[idx[i]] += t*ht[x][n][sj]/marg[j][n][x];
                                 }
                         }
                 }
         }

 }
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `calcD` Never used.

maxD finds the largest value of the directional derivative $D$ – it will be used for error rate estimation, since

$$l(\hat{Q}|\mathbf{X}) - l(Q|\mathbf{X}) \leq \max_{\mathbf{v}} D_Q(\mathbf{v}). \tag{8}$$

"..\src\ReprodCalcs.c" ?≡

```
double maxD(SEXP D, int *idx, int nS){
   int i;
   double currmax, val;

   currmax = 0;
   for (i=0; i<nS; i++){
           val = REAL(D)[idx[i]];
           if (val > currmax) currmax = val;
   }
   return currmax;
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: MaxD Never used.

CalcTopD gets the vectors from $S$ corresponding to the at most ‘limit’ positive D values (that is it calculates $\mathbf{v}_t^1, \ldots, \mathbf{v}_t^m$). The return value is a C-style matrix with each row a non-decreasing vector. The number $m$ of the selected vectors is returned through the pointer nselect.

"..\src\ReprodCalcs.c" ?≡

```
int **CalcTopD(SEXP D, SEXP S, int *idx, int limit, int *nselect, int ntrt, int nS){
         int **res, nmx, i, j, pos, start;
         double *posDvec,  dcut;

           // count the number of non-negative elements in D
     nmx = 0;
     for (i=0; i<nS; i++){
             if (REAL(D)[idx[i]] >= 0){
                     nmx++;
             }
     }
     if (nmx == 0){
             res = 0;
             *nselect = 0;
             return res;
     }

     if (nmx > limit){ //find the limit-th largest D
                 posDvec = malloc(nmx*sizeof(double));
                 pos = 0;
             for (i=0; i<nS; i++){
                     if (REAL(D)[idx[i]] >= 0){
                             posDvec[pos] = -REAL(D)[idx[i]];  //negation is needed, because rPsort uses
                             pos++;
                     }
             }
             rPsort(posDvec, nmx, limit);
             dcut = -posDvec[limit];  //the cutoff for determining the limit-th largest values
             free(posDvec);
     }
     else dcut = 0;

     nmx = imin2(limit, nmx);
```

```
        res = (int**) Calloc(nmx, int*);
        pos = 0;
    for (i=0; i<nS; i++){
            if (pos >= nmx) break;
            if (REAL(D)[idx[i]] < dcut) continue;
            res[pos] =(int*) Calloc(ntrt, int);  //copy the ith row of S
            for (j=0, start=0; j<ntrt; j++, start+=nS) res[pos][j] = INTEGER(S)[start+i];
            pos++;
    }

    *nselect = nmx;
    return res;
}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `CalcTopD` ?.

`NegLogLik` calculates the negative log-likelihood at a potential new mixing distribution as a function of $(\alpha_0, \alpha_1, \ldots, \alpha_m)$, where $\sum_{i=0}^{m} \alpha_i = 1$ and $\alpha_i \geq 0$. To remove the sum-to-one constraint, we reparameterize to $\mathrm{par}_i = \gamma_i = \alpha_i/\alpha_0 \geq 0$, $i = 1, \ldots, m$. With this reparametrization, $\alpha_0 = 1/(1 + \sum \gamma_i)$ and $\alpha_i = \alpha_0\gamma_i$, $i = 1, \ldots, m$, and the constraints are $\gamma_i \geq 0$. The potential new mixing distribution is

$$Q^{\gamma} = \big(Q + \textstyle\sum_i \gamma_i \delta_{\mathbf{v}^i}\big)/(1 + \textstyle\sum_i \gamma_i)$$

The corresponding log-likelihood is

$$l(Q^{\gamma}) = \sum_{g,n,r} A_{gnr} \log Q^{\gamma}_{gn}(r) = \sum_{g,n,r} A_{gnr} \log\big(Q_{gn}(r) + \textstyle\sum_i \gamma_i h(r, v^i_g, n)\big) - N \log(1 + \textstyle\sum_i \gamma_i), \tag{9}$$

where $A_{gnr}$ is the observed number of clusters of size $n$ with $r$ responses in treatment group $g$.

`lmS` stores the $\mathbf{v}^1_t, \ldots, \mathbf{v}^m_t$ vectors calculated by `CalcTopD`. The required variables (ntrt, size, ntot, marg, ht, lmS) will be declared as global and will be available for the procedure, while `tab` will be passed through the `*ex` pointer.

"..\src\ReprodCalcs.c" ?≡

```
⟨ Declare global variables ? ⟩
double NegLogLik(int npar, double *par, void *ex){
         //par[j] = (alpha_(j+1)/alpha_0), j=0,...,nmax-1
   int j, n, r, i, sj, x;
   double res, sum;
   SEXP tab;

   tab = (SEXP)ex;
   res = 0;

   for (j=0; j<ntrt; j++){
           for (n=1; n<=size; n++){
                   for (r=0; r<=n; r++){
                           x = GetTabElem(tab,size,n,r,j);
                           if (x>0){
                                   sum = marg[j][n][r];
                                   for (i=0; i<npar; i++){
                                           sj = lmS[i][j];
                                           sum += par[i]*ht[r][n][sj];
                                   }
                                   res += x*log(sum);
                           }
                   }
```

```
          }
      }
      sum = 0;
      for (i=0; i<npar; i++) sum += par[i];
      res -= ntot*log1p(sum);  //log1p(sum)=log(1+sum)

      if (!R_FINITE(res)){
        res = 1e60;
      }

      return (-res); }
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `NegLogLik` ?.

`NegLogLikDeriv` calculates the gradient vector of the negative log-likelihood function defined above.

$$\frac{\partial l}{\partial \gamma_u} = \sum_{g,n,r} A_{gnr} \frac{h(r, v_g^u, n)}{Q_{gn}(r) + \sum_i \gamma_i h(r, v_g^i, n)} - \frac{N}{1 + \sum_i \gamma_i} \tag{10}$$

`"..\src\ReprodCalcs.c"` ?≡

```
void NegLogLikDeriv(int npar, double *par, double *gr, void *ex){
   int j, n, r, i, sj, x;
   double alpha0, sum, ***denom;
   SEXP tab;

   tab = (SEXP)ex;
   //prepare the shared denominators
   denom = malloc(ntrt*sizeof(double*));
   for (j=0; j<ntrt; j++){
           denom[j] = malloc((size+1)*sizeof(double*));
           for (n=1; n<=size; n++) denom[j][n] = calloc(n+1, sizeof(double));
   }
   for (j=0; j<ntrt; j++){
           for (n=1; n<=size; n++){
                   for (r=0; r<=n; r++){
                           sum = marg[j][n][r];
                           for (i=0; i<npar; i++){
                                   sj = lmS[i][j];
                                   sum += par[i]*ht[r][n][sj];
                           }
                           denom[j][n][r] = sum;
                   }
           }
   }

   alpha0 = 1;
   for (i=0; i<npar; i++)  alpha0 += par[i];
   alpha0 = 1.0/alpha0;

   //calc the gradients
   for (i=0; i<npar; i++){
           sum = -ntot*alpha0;
           for (j=0; j<ntrt; j++){
                   for (n=1; n<=size; n++){
                           for (r=0; r<=n; r++){
                                   x = GetTabElem(tab,size,n,r,j);
                                   if (x>0){
                                           sj = lmS[i][j];
                                           sum += x*ht[r][n][sj]/denom[j][n][r];
                                   }
                           }
                   }
           }
           gr[i] = -sum;
   }

   for (j=0; j<ntrt; j++){
           for (n=1; n<=size; n++) free(denom[j][n]);
           free(denom[j]);
   }
   free(denom);
}
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

`UpdateQ` performs the updating step of the ISDM algorithm once the optimal $\alpha_i$ values have been found (the input parameter `g` has the reparameterized `par` values at the optimum): the mixing distribution is moved toward the values with the largest directional derivative.

$$Q^{(t+1)} = \big(Q + \textstyle\sum_i \gamma_i \delta_{\mathbf{v}^i}\big)/(1 + \textstyle\sum_i \gamma_i) \tag{11}$$

`lmS_idx` is the index vector corresponding to the location in $Q$ of the selected directions.
"..\src\ReprodCalcs.c" ?≡

```
void UpdateQ(SEXP Q, double *g, int nS, int nmax, int *idx, int *lmS_idx){
   double alpha0;
   int i;

   alpha0 = 1;
   for (i=0; i<nmax; i++) alpha0 += g[i];
   alpha0 = 1/alpha0;

   for (i=0; i<nS; i++) REAL(Q)[idx[i]] *= alpha0;
   for (i=0; i<nmax; i++) REAL(Q)[lmS_idx[i]] += alpha0 * g[i];
}
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `UpdateQ` ?.

`UpdateMarginals` updates the marginal distribution calculated by `CalcMarginals` after each iterative step – this is faster than recalculating them again. `lmS` contains the selected directions.

$$Q^{\gamma}_{gn}(r) = \big(Q_{gn}(r) + \textstyle\sum_i \gamma_i f(r, v^i_g, n)\big)\frac{1}{1+\sum_i \gamma_i} \tag{12}$$

"..\src\ReprodCalcs.c" ?≡

```
 void UpdateMarginals(double ***marg, double *g, double ***ht, int **lmS,
                      int ntrt, int size, int nmax){
   double alpha0;
   int i, j, n, r, sj;

    alpha0 = 1;
        for (i=0; i<nmax; i++){
                alpha0 += g[i];
        }
        alpha0 = 1/alpha0;

        for (j=0; j<ntrt; j++){
                for (n=1; n<=size; n++){
                        for (r=0; r<=n; r++){
                                for (i=0; i<nmax; i++){
                                        sj = lmS[i][j];
                                        marg[j][n][r] += g[i] * ht[r][n][sj];
                                }
                                marg[j][n][r] *= alpha0;
                        }
                }
        }
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `UpdateMarginals` ?.

Finally, `ReprodISDM` sets up all the required matrices, and runs the ISDM iterations while monitoring the error. The maximization of the log-likelihood is done by the built-in R function `lbfgsb` which implements the quasi-Newton method of Byrd95 that allows boundary constraints for optimization. It is actually a minimization routine, so the negative likelihood defined above is used. The `MaxDirection` parameter controls the number of directions which are considered at each step of the ISDM algorithm. Values of 0 (or less) mean setting it to the number of non-empty cells in the data. The variables required by the log-likelihood function and its derivative are declared global.

⟨ *Declare global variables* ? ⟩ ≡

```
int ntrt, size, **lmS;
double ntot, ***ht, ***marg;
◇
```

Fragment referenced in ?.

"..\src\ReprodCalcs.c" ?≡

```
SEXP ReprodISDM(SEXP Q, SEXP S, SEXP tab, SEXP MaxIter, SEXP MaxDirections,
                SEXP eps, SEXP verbose){
 SEXP dims, D,  res, margSXP, tmp;
 int i, j, n, r, *idx, nS, niter, nmax, fncount, grcount, fail,
     *boundtype,  limit, *lmS_idx, nenforced;
 double rel_error, *gamma, *lower, *upper, NLLmin;
 char msg[60];


 PROTECT(dims = GET_DIM(tab));
   size = INTEGER(dims)[0];
   ntrt = INTEGER(dims)[2];
 UNPROTECT(1);

 PROTECT(dims = GET_DIM(S));
   nS = INTEGER(dims)[0];
 UNPROTECT(1);


 PROTECT(D = duplicate(Q));
 for (i=0; i<length(Q); i++) REAL(D)[i] = 0;
```

```
idx = IndexVector(S, size+1, ntrt, nS);
ht = HyperTable(size);

ntot=0;
for (i=0; i<length(tab); i++) ntot += REAL(tab)[i];

limit = INTEGER(coerceVector(MaxDirections, INTSXP))[0];
if (limit <= 0){  //set it to the number of non-empty cells
 limit=0;
 for (i=0; i<length(tab); i++){
     if (REAL(tab)[i]>0) limit++;
 }
}

marg = CalcMarginals(S, Q, ht, idx, ntrt, size, nS);
CalcD(D, S, tab, idx, ht, marg, ntrt, nS, size, ntot);

rel_error = maxD(D, idx, nS);
niter = 0;
nenforced = 0;

while (niter < asInteger(MaxIter) && rel_error > asReal(eps)){
    R_CheckUserInterrupt();
    niter++;

    lmS = CalcTopD(D, S, idx, limit, &nmax, ntrt, nS);
    lmS_idx = IndexVectorC(lmS, size+1, ntrt, nmax);

    if (nmax == limit) nenforced++;

    gamma = (double*) Calloc(nmax, double);
    lower = (double*) Calloc(nmax, double);
    upper = (double*) Calloc(nmax, double);
    boundtype = (int*) Calloc(nmax, int);

    for (i=0; i<nmax; i++){
        gamma[i] = 0;
        lower[i] = 0;
        upper[i] = imin2(1e6/nmax, 100); // => alpha0>1e-6
        boundtype[i] = 1; //lower  bound only
    }


    lbfgsb(nmax, 5, gamma, lower, upper, boundtype, &NLLmin, NegLogLik, NegLogLikDeriv,
           &fail, tab, 1e5, 0, &fncount, &grcount, 1000, msg, asInteger(verbose), 10);

    UpdateMarginals(marg, gamma, ht, lmS, ntrt, size, nmax);
 CalcD(D, S, tab, idx, ht, marg, ntrt, nS, size, ntot);
    UpdateQ(Q, gamma, nS, nmax, idx, lmS_idx); //only needed to be able to return Q
    rel_error = maxD(D, idx, nS);
    if (asInteger(verbose)==1)
     Rprintf("Step %d, rel.error=%f, NLL=%f\n", niter, rel_error, NLLmin);


    Free(gamma);
    Free(lower);
    Free(upper);
    Free(boundtype);
```

```
         for (i=0; i<nmax; i++){
                 Free(lmS[i]);
         }
         Free(lmS);
         free(lmS_idx);
 }

 free(idx);
 for(j=0; j<=size; j++){
          for (n=0; n<=size; n++) free(ht[j][n]);
          free(ht[j]);
  }
 free(ht);

 PROTECT(margSXP = allocVector(REALSXP, ntrt*size*(size+1)));
 PROTECT(dims = allocVector(INTSXP,3));
    INTEGER(dims)[0] = size+1;     INTEGER(dims)[1] = size;     INTEGER(dims)[2] = ntrt;
    i = 0;
    for(j=0; j<ntrt; j++){
          for (n=1; n<=size; n++){
                  for (r=0; r<=n; r++){
                          REAL(margSXP)[i] = marg[j][n][r];
                          i++;
                  }
                  for (r=n+1; r<=size; r++){
                          REAL(margSXP)[i] = NA_REAL;
                          i++;
                  }
          }
   }
 dimgets(margSXP, dims);
 UNPROTECT(1);    //dims
 for(j=0; j<ntrt; j++){
          for (n=1; n<=size; n++) free(marg[j][n]);
          free(marg[j]);
  }
 free(marg);

 PROTECT(res = allocVector(VECSXP,5));
  if (asInteger(verbose)==1)
   Rprintf("Limit=%d; %d Iterations; Limit enforced %d times (%4.2f percent)\n",
           limit, niter, nenforced, nenforced*100.0/niter);

  SET_VECTOR_ELT(res, 0, margSXP);
  SET_VECTOR_ELT(res, 1, Q);
  SET_VECTOR_ELT(res, 2, D);
  SET_VECTOR_ELT(res, 3, ScalarReal(-NLLmin));
  PROTECT(tmp = allocVector(REALSXP,2));
   REAL(tmp)[0] = rel_error;
   REAL(tmp)[1] = niter;
  SET_VECTOR_ELT(res, 4, tmp);
 UNPROTECT(4);   //tmp, res, margSXP, D

 return res;

}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

Defines: `ReprodISDM` ?.
Uses: `CalcMarginals` ?, `CalcTopD` ?, `HyperTable` ?, `IndexVectorC` ?, `NegLogLik` ?, `UpdateMarginals` ?, `UpdateQ` ?.

### 3.2. EM estimation.

The `MixReprodQ` function implements the EM-based fitting:

$$Q^{(t+1)}(\mathbf{v}) = \frac{1}{N}\sum_{g,i}\frac{h(r_{g,i}, v_g, n_{g,i})Q^{(t)}(\mathbf{v})}{\sum_{\mathbf{q}} h(r_{g,i}, q_g, n_{g,i})Q^{(t)}(\mathbf{q})}. \tag{13}$$

Note that from (6)

$$Q^{(t+1)}(\mathbf{v}) = Q^{(t)}(\mathbf{v}) + \frac{1}{N}D_Q(\mathbf{v})Q^{(t)}(\mathbf{v}), \tag{14}$$

so we will be able to use the `CalcMarginals` and `CalcD` functions defined earlier.

`UpdateReprodQ` performs the updating step defined in (13).

"..\src\ReprodCalcs.c" ?≡

```
SEXP UpdateReprodQ(SEXP Q, SEXP S, SEXP tab, int size, int ntrt, int nS,
                   double*** ht, int* idx){
 int ntot, i, j, n;
 double ***marg;
 SEXP resobj, D;

 PROTECT(resobj = duplicate(Q));
 for (i=0; i<length(Q); i++) REAL(resobj)[i] = 0;

 ntot=0;
 for (i=0; i<length(tab); i++) ntot += REAL(tab)[i];

 marg = CalcMarginals(S, Q, ht, idx, ntrt, size, nS);

 PROTECT(D = duplicate(Q));
 for (i=0; i<length(Q); i++) REAL(D)[i] = 0;
 CalcD(D, S, tab, idx, ht, marg, ntrt, nS, size, ntot);

// update Q values
 for (i=0; i<length(Q); i++){
   REAL(resobj)[i] = REAL(Q)[i] * (1+REAL(D)[i]/ntot);
 }

 //cleanup
 for(j=0; j<ntrt; j++){
         for (n=1; n<=size; n++) free(marg[j][n]);
         free(marg[j]);
 }
 free(marg);


 UNPROTECT(2);
 return resobj;
}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `UpdateReprodQ` ?.
Uses: `CalcMarginals` ?.

And, finally, `MixReprodQ` performs the actual EM iterations. `eps` controls the precision of the estimate of the log-likelihood. Since

$$l(\hat{Q}|\mathbf{X}) - l(Q^{(t)}|\mathbf{X}) \leq N \frac{Q^{(t+1)}(\mathbf{v}) - Q^{(t)}(\mathbf{v})}{Q^{(t)}(\mathbf{v})}, \tag{15}$$

we control the largest relative change in $Q$ during updating. If the `verbose` option is selected, the current value of the relative change is displayed at every 10th iteration.

"..\src\ReprodCalcs.c" ?≡

```
SEXP MixReprodQ(SEXP Q, SEXP S, SEXP tab, SEXP MaxIter, SEXP eps, SEXP verbose){
 double rel_error, ***ht, ntot, re, ***marg, loglik;
 int niter, size, ntrt, nS, i, j, n, r, *idx, Qlength;
 SEXP D, Qnew, resQ, dims, resobj, tmp, margSXP, Qdims;

 PROTECT(Qdims = GET_DIM(Q));
 Qlength = 1;
 for (i=0; i<length(Qdims); i++) Qlength *= INTEGER(Qdims)[i];

 PROTECT(resQ = duplicate(Q));

 PROTECT(dims = GET_DIM(tab));
   size = INTEGER(dims)[0];
   ntrt = INTEGER(dims)[2];
 UNPROTECT(1);  //dims

 PROTECT(dims = GET_DIM(S));
   nS = INTEGER(dims)[0];
 UNPROTECT(1); //dims

 ntot=0;
 for (i=0; i<length(tab); i++) ntot += REAL(tab)[i];

 idx = IndexVector(S, size+1, ntrt, nS);
 ht = HyperTable(size);

 PROTECT(tmp = allocVector(REALSXP, 2));
 rel_error = 1;
 niter = 0;
 while ((niter<asInteger(MaxIter))&&(rel_error>asReal(eps))){
          R_CheckUserInterrupt();
                niter++;
    PROTECT(Qnew = UpdateReprodQ(resQ, S, tab, size, ntrt, nS, ht, idx));
    rel_error = 0;
    for (i=0; i<length(Qnew); i++){
            if (REAL(resQ)[i]>0){
                    re = ntot*(REAL(Qnew)[i]-REAL(resQ)[i])/REAL(resQ)[i];
                    if (rel_error < re) rel_error = re;
            }
            REAL(resQ)[i] = REAL(Qnew)[i];
    }
    UNPROTECT(1); //Qnew
    if ((asInteger(verbose) == 1)&&(niter%10 == 1)){
            REAL(tmp)[1] = rel_error;
            REAL(tmp)[0] = niter;
        PrintValue(tmp);
     }

  }
 UNPROTECT(1);  //tmp
```

```
//calculate ML estimates for the output
marg = CalcMarginals(S, resQ, ht, idx, ntrt, size, nS);
PROTECT(margSXP = allocVector(REALSXP, ntrt*size*(size+1)));
PROTECT(dims = allocVector(INTSXP,3));
   INTEGER(dims)[0] = size+1;     INTEGER(dims)[1] = size;     INTEGER(dims)[2] = ntrt;
   i = 0;
   for(j=0; j<ntrt; j++){
         for (n=1; n<=size; n++){
                 for (r=0; r<=n; r++){
                         REAL(margSXP)[i] = marg[j][n][r];
                         i++;
                 }
                 for (r=n+1; r<=size; r++){
                         REAL(margSXP)[i] = NA_REAL;
                         i++;
                 }
         }
  }
dimgets(margSXP, dims);
UNPROTECT(1); //dims

PROTECT(D = allocVector(REALSXP, Qlength));
dimgets(D, Qdims);
for (i=0; i<Qlength; i++) REAL(D)[i] = 0;
CalcD(D, S, tab, idx, ht, marg, ntrt, nS, size, ntot);

//calculate log-likelihood for the output
loglik = 0;
for(j=0; j<ntrt; j++){
 for (n=1; n<=size; n++){
         for (r=0; r<=n; r++){
           loglik += GetTabElem(tab, size, n, r, j)*log(marg[j][n][r]);
    }
  }
 }

for(j=0; j<ntrt; j++){
         for (n=1; n<=size; n++) free(marg[j][n]);
         free(marg[j]);
 }
free(marg);


for(j=0; j<=size; j++){
         for (n=0; n<=size; n++) free(ht[j][n]);
         free(ht[j]);
 }
free(ht);
free(idx);

PROTECT(resobj = allocVector(VECSXP, 5));
  SET_VECTOR_ELT(resobj, 0, margSXP);
  SET_VECTOR_ELT(resobj, 1, resQ);
  SET_VECTOR_ELT(resobj, 2, D);
  SET_VECTOR_ELT(resobj, 3, ScalarReal(loglik));
        PROTECT(tmp = allocVector(REALSXP,2));
          REAL(tmp)[0] = rel_error;
          REAL(tmp)[1] = niter;
```

```
            SET_VECTOR_ELT(resobj, 4, tmp);
    UNPROTECT(6); //tmp, resobj,D, margSXP, resQ, Qdims

    return resobj;

   }

   ◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: `CalcMarginals` ?, `HyperTable` ?, `UpdateReprodQ` ?.

## 4. Testing for stochastic ordering

Now that we have the MLEs, we implement the likelihood-ratio test.

**4.1. Testing against a global null.** First we consider testing

$$H_0 : \boldsymbol{\pi}_1 = \cdots = \boldsymbol{\pi}_G \quad \text{versus} \quad H_a : \boldsymbol{\pi}_1 \preceq^{st} \cdots \preceq^{st} \boldsymbol{\pi}_G$$

We incorporate the `turn` parameter to allow fitting and testing umbrella orderings.

The `SO.LRT` computes the likelihood-ratio test statistic.

"../R/Reprod.R" ?≡

```
   #'Likelihood-ratio test statistic
   #'
   #'\code{SO.LRT} computes the likelihood ratio test statistic for stochastic
   #'ordering against equality assuming marginal compatibility for both
   #'alternatives. Note that this statistic does not have a
   #'\eqn{\chi^2}{chi-squared} distribution, so the p-value computation is not
   #'straightforward. The \code{\link{SO.trend.test}} function implements a
   #'permutation-based evaluation of the p-value for the likelihood-ratio test.
   #'
   #'@export
   #'@param cbdata a \code{CBData} object
   #'@param control an optional list of control settings, usually a call to
   #'\code{\link{soControl}}.  See there for the names of the settable control
   #'values and their effect.
   #'@return The value of the likelihood ratio test statistic is returned with two
   #'attributes:
   #'@return \item{ll0}{the log-likelihood under \eqn{H_0}{H0} (equality)}
   #'@return \item{ll1}{the log-likelihood under \eqn{H_a}{Ha} (stochastic order)}
   #'@author Aniko Szabo
   #'@seealso \code{\link{SO.trend.test}}, \code{\link{soControl}}
   #'@keywords htest nonparametric
   #'@examples
   #'
   #'data(shelltox)
   #'LRT <- SO.LRT(shelltox, control=soControl(max.iter = 100, max.directions = 50))
   #'LRT
   #'
   ◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: `SO.trend.test` ?, `soControl` ?.

"../R/Reprod.R" ?≡

```
SO.LRT <- function(cbdata, control=soControl()){
        # LL under null hypothesis of equality (+ reproducibility)
        a <- with(cbdata, aggregate(Freq, list(ClusterSize=ClusterSize,NResp=NResp), sum))
        names(a)[names(a)=="x"] <- "Freq"
        a$ClusterSize <- as.integer(as.character(a$ClusterSize))
        a$NResp <- as.integer(as.character(a$NResp))
        a$Trt <- 1
    class(a) <- c("CBData", "data.frame")

        b <- mc.est(a)
  b <- merge(cbdata, b, all.x=TRUE, by=c("ClusterSize","NResp"))
  ll0 <- with(b, sum(Freq*log(Prob)))

  # LL under alternative hypothesis of stoch ordering (+ reproducibility)
  res <- SO.mc.est(cbdata, control=control)
  ll1 <- attr(res, "loglik")
  lrt <- 2*(ll1 - ll0)
  attr(lrt, "ll0") <- ll0
  attr(lrt, "ll1") <- ll1
  lrt
 }
```

◇
File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: `mc.est` ?, `SO.mc.est` ?, `soControl` ?.

To get the p-value for the LRT, we use a permutation testing approach. The `SO.trend.test` function relies on the `boot` library for this, thus both the permutation test is straightforward to implement. `R` specifies the number of resamples, `method` could be either "ISDM" or "EM", and `eps` is the precision of the LRT estimate.

"../R/Reprod.R" ?≡

```
#'Likelihood ratio test of stochastic ordering
#'
#'Performs a likelihood ratio test of stochastic ordering versus equality using
#'permutations to estimate the null-distribution and the p-value.  If only the
#'value of the test statistic is needed, use \code{\link{SO.LRT}} instead.
#'
#'The test is valid only under the assumption that the cluster-size
#'distribution does not depend on group. During the estimation of the
#'null-distribution the group assignments of the clusters are permuted keeping
#'the group sizes constant; the within-group distribution of the cluster-sizes
#'will vary randomly during the permutation test.
#'
#'The default value of \code{R} is probably too low for the final data
#'analysis, and should be increased.
#'
#'@import boot
#'@export
#'@param cbdata a \code{\link{CBData}} object.
#'@param R an integer -- the number of random permutations for estimating the
#'null distribution.
#'@param control an optional list of control settings, usually a call to
#'\code{\link{soControl}}.  See there for the names of the settable control
#'values and their effect.
#'@return A list with the following components
```

```
#'@return \item{LRT}{the value of the likelihood ratio test statistic. It has two
#'attributes: \code{ll0} and \code{ll1} - the values of the log-likelihood
#'under \eqn{H_0}{H0} and \eqn{H_a}{Ha} respectively.}
#'@return \item{p.val}{the estimated one-sided p-value.}
#'@return \item{boot.res}{an object of class "boot" with the detailed results of
#'the permutations.  See \code{\link[boot]{boot}} for details.}
#'@author Aniko Szabo, aszabo@@mcw.edu
#'@seealso \code{\link{SO.LRT}} for calculating only the test statistic,
#'\code{\link{soControl}}
#'@references Szabo A, George EO. (2010) On the Use of Stochastic Ordering to
#'Test for Trend with Clustered Binary Data. \emph{Biometrika} 97(1), 95-108.
#'@keywords htest nonparametric
#'@examples
#'
#'\donttest{
#'data(shelltox)
#'set.seed(45742)
#'sh.test <- SO.trend.test(shelltox, R=10, control=soControl(eps=0.1, max.directions=25))
#'sh.test
#'
#'#a plot of the resampled LRT values
#'#would look better with a reasonable value of R
#' null.vals <- sh.test$boot.res$t[,1]
#' hist(null.vals, breaks=10,  freq=FALSE, xlab="Test statistic", ylab="Density",
#'      main="Simulated null-distribution", xlim=range(c(0,20,null.vals)))
#' points(sh.test$LRT, 0, pch="*",col="red", cex=3)
#'}
#'
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: SO.trend.test ?, soControl ?.

"../R/Reprod.R" ?≡

```
SO.trend.test <- function(cbdata, R=100, control=soControl()){
        dat2 <- cbdata[rep(1:nrow(cbdata), cbdata$Freq),]  #each row is one sample
        dat2$Freq <- NULL

        boot.LRT.fun <- function(dat, idx){
          dat.new <- cbind(dat[idx, c("ClusterSize","NResp")], Trt=dat$Trt)   #rearrange clusters
                dat.f <- aggregate(dat.new$Trt,
                          list(Trt=dat.new$Trt, ClusterSize=dat.new$ClusterSize, NResp=dat.new$NResp), le
          names(dat.f)[names(dat.f)=="x"] <- "Freq"
    dat.f$ClusterSize <- as.numeric(as.character(dat.f$ClusterSize))
          dat.f$NResp <- as.numeric(as.character(dat.f$NResp))
    class(dat.f) <- c("CBData", class(dat.f))

    stat <- SO.LRT(dat.f, control=control)
    stat}

   res <- boot(dat2, boot.LRT.fun, R=R, sim="permutation")

   p <- mean(res$t[,1] >= res$t0)
   LRT <- res$t0
   list(LRT=LRT, p.val=p, boot.res=res)}
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: SO.trend.test ?, ?, ?, ?.

Uses: soControl ?.

4.2. **Universal trend test function.** trend.test provides a common interface for the Rao-Scott, GEE, and stochastic order based trend tests.

"../R/Reprod.R" ?≡

```
#'Test for increasing trend with correlated binary data
#'
#'The \code{trend.test} function provides a common interface to the trend tests
#'implemented in this package: \code{\link{SO.trend.test}},
#'\code{\link{RS.trend.test}}, and \code{\link{GEE.trend.test}}. The details of
#'each test can be found on their help page.
#'
#'@export
#'@param cbdata a \code{\link{CBData}} object
#'@param test character string defining the desired test statistic. "RS"
#'performs the Rao-Scott test (\code{\link{RS.trend.test}}), "SO" performs the
#'stochastic ordering test (\code{\link{SO.trend.test}}), "GEE", "GEEtrend",
#'"GEEall" perform the GEE-based test (\code{\link{GEE.trend.test}}) with
#'constant, linearly modeled, and freely varying scale parameters,
#'respectively.
#'@param exact logical, should an exact permutation test be performed. Only an
#'exact test can be performed for "SO". The default is to use the asymptotic
#'p-values except for "SO".
#'@param R integer, number of permutations for the exact test
#'@param control an optional list of control settings for the stochastic order
#'("SO") test, usually a call to \code{\link{soControl}}.  See there for the
#'names of the settable control values and their effect.
#'@return A list with two components and an optional "boot" attribute that
#'contains the detailed results of the permutation test as an object of class
#'\code{\link[boot]{boot}} if an exact test was performed.
#'@return \item{statistic}{numeric, the value of the test statistic}
#'@return \item{p.val}{numeric, asymptotic one-sided p-value of the test}
#'@author Aniko Szabo, aszabo@@mcw.edu
#'@seealso \code{\link{SO.trend.test}}, \code{\link{RS.trend.test}}, and
#'\code{\link{GEE.trend.test}} for details about the available tests.
#'@keywords htest nonparametric
#'@examples
#'
#'data(shelltox)
#'trend.test(shelltox, test="RS")
#'set.seed(5724)
#'#R=50 is too low to get a good estimate of the p-value
#'trend.test(shelltox, test="RS", R=50, exact=TRUE)
#'
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: SO.trend.test ?, soControl ?.

"../R/Reprod.R" ?≡

```
trend.test <- function(cbdata, test=c("RS","GEE","GEEtrend","GEEall","SO"), exact=test=="SO",
                       R=100, control=soControl()){
   test <- match.arg(test)
   if (!exact && !(test=="SO")){
     res <- switch(test, RS=RS.trend.test(cbdata),
                         GEE=GEE.trend.test(cbdata,scale.method="fixed"),
                         GEEtrend=GEE.trend.test(cbdata,scale.method="trend"),
                         GEEall=GEE.trend.test(cbdata,scale.method="all"))
   }
   else {
     dat2 <- cbdata[rep(1:nrow(cbdata), cbdata$Freq),]  #each row is one sample
     dat2$Freq <- NULL

     boot.LRT.fun <- function(dat, idx){
       dat.new <- cbind(dat[idx, c("ClusterSize","NResp")], Trt=dat$Trt)   #rearrange clusters
       dat.f <- aggregate(dat.new$Trt,
                  list(Trt=dat.new$Trt, ClusterSize=dat.new$ClusterSize, NResp=dat.new$NResp), length)
       names(dat.f)[names(dat.f)=="x"] <- "Freq"
       dat.f$ClusterSize <- as.numeric(as.character(dat.f$ClusterSize))
       dat.f$NResp <- as.numeric(as.character(dat.f$NResp))
       class(dat.f) <- c("CBData", class(dat.f))

       stat <- switch(test, SO=SO.LRT(dat.f, control=control),
                            RS=RS.trend.test(dat.f)$statistic,
                            GEE=GEE.trend.test(dat.f, scale.method="fixed")$statistic,
                            GEEtrend=GEE.trend.test(cbdata,scale.method="trend")$statistic,
                            GEEall=GEE.trend.test(cbdata,scale.method="all")$statistic)
       stat}

     bootres <- boot(dat2, boot.LRT.fun, R=R, sim="permutation")
     res <- list(statistic=bootres$t0, p.val= mean(bootres$t[,1] >= bootres$t0))
     attr(res, "boot") <- bootres
   }
   res}
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: soControl ?.

4.3. **Finding the NOAEL.** The NOSTASOT dose is the No-Statistical-Significance-Of-Trend dose – the largest dose at which no trend in the rate of adverse events has been observed.

"../R/Reprod.R" ?≡

```
#'Finding the NOSTASOT dose
#'
#'The NOSTASOT dose is the No-Statistical-Significance-Of-Trend dose -- the
#'largest dose at which no trend in the rate of response has been observed. It
#'is often used to determine a safe dosage level for a potentially toxic
#'compound.
#'
#'A series of hypotheses about the presence of an increasing trend overall,
#'with all but the last group, all but the last two groups, etc.  are tested.
#'Since this set of hypotheses forms a closed family, one can test these
#'hypotheses in a step-down manner with the same \code{sig.level} type I error
#'rate at each step and still control the family-wise error rate.
#'
#'The NOSTASOT dose is the largest dose at which the trend is not statistically
#'significant. If the trend test is not significant with all the groups
#'included, the largest dose is the NOSTASOT dose. If the testing sequence goes
#'down all the way to two groups, and a significant trend is still detected,
#'the lowest dose is the NOSTASOT dose. This assumes that the lowest dose is a
#'control group, and this convention might not be meaningful otherwise.
#'
```

```
#'@export
#'@param cbdata a \code{\link{CBData}} object
#'@param test character string defining the desired test statistic. See
#'\code{\link{trend.test}} for details.
#'@param exact logical, should an exact permutation test be performed. See
#'\code{\link{trend.test}} for details.
#'@param R integer, number of permutations for the exact test
#'@param sig.level numeric between 0 and 1, significance level of the test
#'@param control an optional list of control settings for the stochastic order
#'("SO") test, usually a call to \code{\link{soControl}}.  See there for the
#'names of the settable control values and their effect.
#'@return a list with two components
#'@return \item{NOSTASOT}{character string identifying the NOSTASOT dose.}
#'@return \item{p}{numeric vector of the p-values of the tests actually performed.}
#'The last element corresponds to all doses included, and will not be missing.
#'p-values for tests that were not actually performed due to the procedure
#'stopping are set to NA.
#'@author Aniko Szabo, aszabo@@mcw.edu
#'@seealso \code{\link{trend.test}} for details about the available trend
#'tests.
#'@references Tukey, J. W.; Ciminera, J. L. & Heyse, J. F. (1985) Testing the
#'statistical certainty of a response to increasing doses of a drug.
#'\emph{Biometrics} 41, 295-301.
#'@keywords htest nonparametric
#'@examples
#'
#'data(shelltox)
#'NOSTASOT(shelltox, test="RS")
#'
◇
```

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Uses: NOSTASOT ?, soControl ?.

"../R/Reprod.R" ?≡

```
NOSTASOT <- function(cbdata, test=c("RS","GEE","GEEtrend","GEEall","SO"), exact=test=="SO",
                     R=100, sig.level=0.05, control=soControl()){
   ntrt <- nlevels(cbdata$Trt)
   control.gr <- levels(cbdata$Trt)[1]
   p.vec <- array(NA, ntrt-1)
   names(p.vec) <- levels(cbdata$Trt)[-1]
   NOSTASOT.found <- FALSE
   curr.gr.idx <- ntrt
   curr.gr <- levels(cbdata$Trt)[ntrt]

   while (!NOSTASOT.found & (curr.gr.idx>1)){
     d1 <- cbdata[unclass(cbdata$Trt)<=curr.gr.idx, ]
     d1$Trt <- factor(d1$Trt) #eliminate unused levels
     tr.test <- trend.test(d1, test=test, exact=exact, R=R, control=control)
     p.vec[curr.gr] <- tr.test$p.val
     if (tr.test$p.val < sig.level){ #NOSTASOT not found yet
       curr.gr.idx <- curr.gr.idx - 1
       curr.gr <- levels(cbdata$Trt)[curr.gr.idx]
     }
     else { #NOSTASOT
       NOSTASOT.found <- TRUE
     }
   }
```

```
        list(NOSTASOT = curr.gr, p=p.vec)
    }
```

◇

File defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
Defines: `NOSTASOT` ?.
Uses: `soControl` ?.

## 5. Files

`"../R/aaa-generics1.R"` Defined by ?, ?.
`"../R/Reprod.R"` Defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.
`"..\src\ReprodCalcs.c"` Defined by ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?.

## 6. Macros

⟨Combine the two parts of the sequences ?⟩ Referenced in ?.
⟨Convert 'a' to non-decreasing sequence and insert into 'res' ?⟩ Referenced in ?.
⟨Declare global variables ?⟩ Referenced in ?.
⟨Generate non-decreasing sequences of length ntrt-turn with values between sq and size ?⟩ Referenced in ?.
⟨Generate non-increasing sequences of length turn with values ≤ size ?⟩ Referenced in ?.
⟨Take care of turn=1 and turn=ntrt ?⟩ Referenced in ?.

## 7. Identifiers