# Resampling-Based Multiple Hypothesis Testing with Applications to Genomics: New Developments in the R/Bioconductor Package multtest

Houston N. Gilbert[*] Katherine S. Pollard[†]
Mark J. van der Laan[‡] Sandrine Dudoit[**]

[*]University of California - Berkeley, houston@stat.berkeley.edu
[†]University of California, San Francisco, kpollard@gladstone.ucsf.edu
[‡]University of California - Berkeley, laan@berkeley.edu
[**]Division of Biostatistics and Department of Statistics, University of California, Berkeley, sandrine@stat.berkeley.edu

# Resampling-Based Multiple Hypothesis Testing with Applications to Genomics: New Developments in the R/Bioconductor Package multtest

Houston N. Gilbert, Katherine S. Pollard, Mark J. van der Laan, and Sandrine Dudoit

**Abstract**

The **multtest** package is a standard Bioconductor package containing a suite of functions useful for executing, summarizing, and displaying the results from a wide variety of multiple testing procedures (MTPs). In addition to many popular MTPs, the central methodological focus of the **multtest** package is the implementation of powerful *joint multiple testing procedures*. Joint MTPs are able to account for the dependencies between test statistics by effectively making use of (estimates of) the test statistics joint null distribution. To this end, two additional bootstrap-based estimates of the test statistics joint null distribution have been developed for use in the package. For asymptotically linear estimators involving single-parameter hypotheses (such as tests of means, regression parameters, and correlation parameters using *t*-statistics), a computationally efficient joint null distribution estimate based on influence curves is now also available. New MTPs implemented in **multtest** include marginal adaptive procedures for control of the false discovery rate (FDR) as well as empirical Bayes joint MTPs which can control any Type I error rate defined as a function of the numbers of false positives and true positives. Examples of such error rates include, among others, the family-wise error rate and the FDR. S4 methods are available for objects of the new class *EBMTP*, and particular attention has been given to reducing the need for repeated resampling between function calls.

## 1 Introduction

Multiple hypothesis testing has statistical applications in fields such as genomics, astronomy, finance, as well as many other settings in which a large number of variables are measured on each subject. The **multtest** package [Pollard et al., 2005] was developed as part of the Bioconductor project [Gentleman et al., 2004, `http://www.bioconductor.org`], with particular emphasis given to the analysis of continous gene expression outcomes such as those obtained in microarray experiments. Multiple testing problems in genomics settings are characterized by working with large multivariate data generating distributions $P$ with unknown dependencies between variables. In addition to including a collection of marginal FWER- and FDR-controlling multiple testing procedures (MTPs) – for example the Bonferroni [Bonferroni, 1936] procedure for control of the family-wise error rate (FWER) or the Benjamini-Hochberg [BH; Benjamini and Hochberg, 1995] procedure for control of the false disocvery rate (FDR) – the main methodological focus in **multtest** has been on the software implementation of *joint multiple testing procedures*. Joint MTPs incorporate information about the dependencies between test statistics into the hypothesis decision-making process. As a result, joint MTPs are often more powerful than their marginal MTP counterparts.

For any choice of MTP, specification of the test statistics (joint) distribution is crucial in order to yield cut-offs, rejection regions and adjusted $p$-values which probabilistically control a Type I error rate. Common choices of null distribution have focused on the permutation distribution or the use of a (null-restricted) bootstrap based on a data generating null distribution $P_0$ [Westfall and Young, 1993, Churchill and Doerge, 1994, Yekutieli and Benjamini, 1999, Tusher et al., 2001, Tibshirani et al., 2001]. Such methods can imply strong statements regarding the parameters of the corresponding null hypotheses – e.g., (marginal) independence of an outcome as measured between two groups versus simply a difference in mean – and may rely on restrictive assumptions such as *subset pivotality* [Westfall and Young, 1993]. Dudoit and van der Laan [2008, Chapter 2] provide a more general characterization of the test statistics null distribution based on *null domination conditions*, in which one selects a test statistics null distribution $Q_0$ (or estimator thereof $Q_{0n}$) that stochastically dominates the true distribution of the test statistics $Q_n = Q_n(P)$. This framework, with attention focused on $Q_n = Q_n(P)$ rather than on a distribution implied by $P_0$, has led to the formulation of several other choices for null distributions.

The first original proposal of Pollard and van der Laan [2004], Dudoit et al. [2004], and van der Laan et al. [2004b], defines the null distribution as the asymptotic distribution of a vector of *null shift and scale-transformed test statistics*, based on user-supplied upper bounds for the means and variances of the test statistics for the true null hypotheses [Dudoit and van der Laan, 2008, Section 2.3]. A simple alternative to the 'centered and scaled' null distribution is the distribution of *null shift-transformed test statistics*, in which the scaling parameters from the former transformation have been removed. A third choice of test statistics joint null distribution is that of van der Laan and Hubbard [2006], who define the null distribution as the asymptotic distribution of a vector of *null quantile-transformed test statistics* [Dudoit and van der Laan, 2008, Section 2.4]. Bootstrap-based estimators of all three null distributions are now available in the **multtest** package.

For a broad class of testing problems, such as the test of $M$ single-parameter null hypotheses using $t$-statistics, an asymptotically valid null distribution is the $M$-variate Gaussian distribution $\mathrm{N}(0, \sigma^*)$, with mean vector zero and covariance matrix $\sigma^* = \Sigma^*(P)$ equal to the *correlation matrix of the vector influence curve* for the parameter of interest [Pollard and van der Laan, 2004, Gilbert et al., 2009, Dudoit and van der Laan, 2008, Section 2.6]. In this case, one may simulate from suitable multivariate normal distribution rather than calculating permutation- or bootstrap-based test statistics. Together with improvements made to earlier versions of **multtest** [Taylor et al., 2007], which include the use of the **snow** package [Tierney et al., 2006] for farming out the resampling operations to nodes on a cluster, the addition of test statistics null distributions based on influence curves can reduce computational bottlenecks associated with resampling-based procedures.

Several MTPs – e.g., FWER-controlling minP or maxT procedures, or the FDR-controlling BH procedure [Benjamini and Hochberg, 1995] – assume all hypotheses belong to the set of true null hypotheses $\mathcal{H}_0$. It has been noted that the BH procedure becomes conservative at a rate proportional to the number of true null hypotheses $h_0 = |\{\mathcal{H}_0\}|$ [Benjamini and Hochberg, 2000, Efron et al., 2001, Storey, 2002, Storey and Tibshirani, 2003, Benjamini et al., 2006]. That is, for a test at nominal level $\alpha = 0.05$, if the proportion of true null hypotheses $h_0/M$ is equal to 0.75, the BH procedure will control the FDR at actual level 0.375. By obtaining an estimate $h_{0n}$ of the number of true null hypotheses $h_0$, *adaptive linear step-up FDR-controlling procedures* attempt to overcome this conservativeness by applying a multiplicative correction factor to the original BH procedure. The adaptive Benjamini-Hochberg [ABH; Benjamini and Hochberg, 2000] and the two-stage Benjamini-Hochberg [TSBH; Benjamini et al., 2006] procedures were further characterized in Dudoit et al. [2008], and they have now been included in the `mt.rawp2adjp` function which returns adjusted $p$-values for marginal MTPs.

*Empirical Bayes (joint) multiple testing procedures* (EBMTPs) represent another approach to Type I error control [van der Laan et al., 2005, Dudoit et al., 2008, Dudoit and van der Laan, 2008, Chapter 7, Procedure 7.1]. EBMTPs may be implemented for *any* tail probability or expected value Type I error rate which can be expressed a function $g(V_n, S_n)$ of the number of false positives $V_n$ and true positives $S_n$. Examples of such error rates include not only the FWER ($g(V_n) = \mathsf{Pr}(V_n > 0)$) and the FDR ($g(V_n, S_n) = V_n/(V_n + S_n)$), but also other Type I error rates such as the *generalized family-wise error rate* (gFWER) for controlling the probability of $k+1$ or more false positives, i.e., $\mathsf{Pr}(V_n > k) \leq \alpha$, or the *tail probability of the proportion of false positives* (TPPFP) for controlling a bound $q$ on the false discovery proportion, i.e., $\mathsf{Pr}(V_n/(V_n + S_n) > q) = \mathsf{Pr}(V_n/R_n > q) \leq \alpha$, where $R_n$ denotes the total number of rejected hypotheses. EBMTPs for control of the FWER, gFWER, TPPFP and FDR have been implemented in the new user-level function `EBMTP`. Methods for manipulating, summarizing, and plotting the results of the new *EBMTP* class objects have also been written for the **multtest** package.

We will elaborate further on the recent developments in **multtest**, highlighting the additions to our software with an application to a publicly available microarray dataset taken from Chiaretti et al. [2004, **ALL** experimental data package, `http://www.bioconductor.org`]. For reproducibility purposes, this document will largely be generated using the `Sweave` function [Leisch, 2002] from the R **tools** package [R Development Core Team, 2009]. Due to the dimensionality of the filtered **ALL** dataset of Chiaretti et al. [2004], however, it will be necessary at times to (i) restrict portions of the analysis to a smaller subset of genes, (ii) use fewer rounds of (re)sampling, and/or (iii) load the output of previously executed code in order to illustrate the utility of our package. It is our intent to make the reader clear of when any of these cases occur. All code, output files, and stored data objects are available in the supplementary material of this paper.

## 2 multtest basics

The **multtest** package has earlier supporting documentation in the form of vignettes and help files which are available to the user from the package directories, the Bioconductor website (`http://www.bioconductor.org`), or from within an active R session. The purpose of this section is to reintroduce the reader to analysis options available in the main user-level functions `MTP` and `EBMTP` (to be discussed below). The MTP function, which contains options for conducting resampling-based multiple hypothesis testing as well as for controlling the output from such procedures, has several function arguments.

```
> library(multtest)
> args(MTP)

function (X, W = NULL, Y = NULL, Z = NULL, Z.incl = NULL, Z.test = NULL,
    na.rm = TRUE, test = "t.twosamp.unequalvar", robust = FALSE,
    standardize = TRUE, alternative = "two.sided", psi0 = 0,
    typeone = "fwer", k = 0, q = 0.1, fdr.method = "conservative",
    alpha = 0.05, smooth.null = FALSE, nulldist = "boot.cs",
    B = 1000, ic.quant.trans = FALSE, MVN.method = "mvrnorm",
    penalty = 1e-06, method = "ss.maxT", get.cr = FALSE, get.cutoff = FALSE,
    get.adjp = TRUE, keep.nulldist = TRUE, keep.rawdist = FALSE,
    seed = NULL, cluster = 1, type = NULL, dispatch = NULL, marg.null = NULL,
    marg.par = NULL, keep.margpar = TRUE, ncp = NULL, perm.mat = NULL,
    keep.index = FALSE, keep.label = FALSE)
NULL
```

The most important considerations are the types of data objects which **multtest** supports. Arguments for specifying these objects are the first entries in the MTP function definition, and they are summarized in Table 1. A variety of test statistics have also been implemented in **multtest**, many of which come with out-of-the-box robust alternatives (robust=TRUE), or options for using nonstandardized *difference statistics* (standardize=FALSE). A summary of available test statistics and the null distributions supported by those choices of test statistics are in Table 2.

The package **multtest** uses closures in the functions MTP and EBMTP to compute test statistics. These closures are defined in terms of attributes which describe each choice of test statistic (see, e.g., the help file corresponding to meanX). The closure, written in R, may be called at two separate stages in the analysis. This closure is used once by the function get.Tn to compute a vector of observed test statistics, and then possibly again by the function boot.null when computing bootstrap test statistics. In this case, the closure is eventually given to an internal function bootloop, which performs the bootstrap calculations in C. In either case, the closure returns the test statistics in a form which allows for the handling of sidedness (e.g., alternative=c('two.sided', 'greater', 'less')) and standardization options. Specifically, the observed test statistics are stored in a matrix obs with numerator in the first row (possibly absolute value or negative, depending on the value of alternative), denominator in the second row, and a 1 or -1 in the third row (again, depending on the value of alternative). The vector of observed test statistics is obs[1,]*obs[3,]/obs[2,].

One exception to the closure rule was made in the case of tests of correlation parameters, which are also new to the **multtest** package. Given a $J \times n$ matrix in X with $J$ variables and $n$ observations, there are $M = \binom{J}{2} = J(J-1)/2$ hypotheses corresponding to all pairwise combinations of variables in X. This case distinguishes itself from all other previously implemented test statistics where the number of hypotheses corresponded to nrow(X), i.e., $J = M$. In this case, a 'closure-like' function corr.Tn was created to calculate test statistics for hypotheses involving correlation parameters and to return the test statistics in a form, namely a matrix obs as above, which could then be used by the rest of **multtest** functionality. Because no formal closure was written for when test='t.cor' or test='z.cor', only null distributions derived from influence curves (nulldist='ic', see below) are currently available for testing hypotheses involving correlation parameters. Implementing resampling-based null distributions for correlation parameters is future work.

A central methodological motivation for initially writing the `MTP` function was the development of powerful joint MTPs for control of the FWER. To this end, a variety of methods for controlling the FWER were made available to the user through the `methods` argument in `MTP`. These options include `'ss.maxT'`, `'sd.maxT'`, `'ss.minP'`, and `'sd.minP'`, which correspond to *single-step* and *step-down* procedures based on maximum test statistics or minimum $p$-values taken over (subsets of) the hypotheses over `B` rounds of resampling. The `typeone` argument has options for control of not only the FWER (`'fwer'`), but also the gFWER, TPPFP, and FDR (`'gfwer'`, `'tppfp'`, and `'fdr'`). Control of these relatively more complicated error rates was obtained through *augmentation multiple testing procedures* [AMTPs; van der Laan et al., 2004a, Dudoit and van der Laan, 2008, Chapter 6]. AMTPs use the results of a FWER-controlling MTP and transform or *augment* those results in a way which guarantees Type I error control of the other desired error rate. AMTPs, while mathematically elegant, have been shown to be conservative in practice. This observation was a motivation for the development of the EBMTPs (described below), which seek to control a given Type I error rate more directly, rather than through the augmentation of a FWER-controlling procedure.

The functions `MTP` and `EBMTP` return objects of class *MTP* and *EBMTP*, respectively. Each class definition contains object slots with relevant information regarding the corresponding MTP. For the purposes of this paper, the most important slots common to objects of both classes are in Table 3. Further slots specific to objects of class *EBMTP* will be introduced in the sections that follow.

Finally, S4 methods have been written to work with objects of both the *MTP* and *EBMTP* classes. These include the methods `print`, `summary`, `plot`, `as.list`, and `'['`. The `plot` methods produces anywhere from four to six different plots for summarizing MTP results and exploring various diagnostic quality checks. The `as.list` method will convert an *MTP* or *EBMTP* class object into a list, while the subsetting method `'['` will subset all multidimensional slot objects and return the MTP results specific to particular selected hypotheses. Because of the differences in how vanilla MTPs and EBMTPs control the Type I error rate, separate updating methods have been written for objects of each class (`update` and `EBupdate`). The details of these methods are given in the `MTP-methods` (alias `EBMTP-methods`) help file and will also be presented as needed below.

## 3 New developments and software additions

The purpose of this section is to detail the implementation of our methods (i) for obtaining consistent estimators of the test statistics joint null distribution and (ii) for conducting marginal adaptive FDR-controlling MTPs as well as joint EBMTPs for control of generalized Type I error rates. The statistical underpinnings of each of these methods has been previously described elsewhere, most recently in van der Laan et al. [2005], van der Laan and Hubbard [2006], Benjamini et al. [2006], Dudoit et al. [2008], Dudoit and van der Laan [2008], and Gilbert et al. [2009]. Code examples demonstrating the **multtest** user interface employ the ALL microarray dataset of Chiaretti et al. [2004]. A full description of the dataset and preprocessing is given in Section 4.1. For now, gene expression measures are stored in a 2051 genes $\times$ $n = 79$ patients matrix `X`, whereas numeric class labels indicating group membership between ALL patients with a particular phenotype and those without are given in the vector `Ynum`. We are interested in testing for differential gene expression between the two groups.

### 3.1 Test statistics joint null distributions

#### 3.1.1 Bootstrap-based null distributions

Bootstrap estimation of the test statistics null distribution in **multtest** occurs by transforming the original test statistics bootstrap distribution in such a way that particular *null domination conditions* are satisified [Dudoit and van der Laan, 2008, Chapter 2]. Previously, only the bootstrap-based estimator of the *null shift and scale-transformed test statistics null distribution* had been included in **multtest** [Dudoit and van der Laan, 2008, Procedure 2.3, `nulldist='boot.cs'`]. Here, the elements of the matrix of bootstrap test statistics $\mathbf{T}_n^B$ are row-**c**entered and **s**caled depending on values dictated by the choice of test statistic. It may be the case, however that the scaling step contributes additional variance to the estimate of the test statistics null distribution [Dudoit and van der Laan, 2008, Taylor and Pollard, 2009]. In this situation, one may wish to only row-center the matrix of bootstrap test statistics. A procedure for obtaining an estimator of the *null shift-transformed test statistics null distribution* (`nulldist='boot.ctr'`) therefore proceeds as before but with the scaling steps removed.

Alternatively, one may also use the bootstrap to estimate the test statistics null distribution as the asymptotic distribution of an $M$-vector of *null quantile-transformed test statistics* [van der Laan and Hubbard, 2006, Dudoit and van der Laan, 2008, Procedure 2.4, `nulldist='boot.qt'`]. Specifically, this procedure proposes the use of the bootstrap to estimate the test statistics correlation structure, while also subsequently imposing an user-specified test statistics marginal distribution on the final estimate. The central intuitive advantage to working with this particular null distribution is that one preserves the correlation structure among test statistics while also using the marginal distribution one would have typically used in the univariate testing scenario. Marginal null distributions may be test statistic-specific (e.g., $z$-statistics, $t$-statistics, $F$-statistics, etc.), or they may be based on a data generating null distribution $P_0$. A more advanced application of the null quantile-transformed null distribution, for example, may include using marginal null distributions obtained from permutations, i.e., leveraging the bootstrap to estimate the dependencies between test statistics corresponding to hypotheses of marginal independence. Because the use of bootstrap null distributions is based on asymptotic results, parametric or permutation-based marginal distributions often perform better with small numbers of samples [Pollard and van der Laan, 2004, van der Laan and Hubbard, 2006].

Bootstrap resampling is performed in **multtest** via an internal call to the function `boot.null`. This function has many (formatted) values passed to it based on the arguments specified in the original `MTP` or `EBMTP` function call. At the heart of the code in `boot.null` is the function `boot.resample`, which uses compiled C code (`bootloop`) to more efficiently apply the R-language test statistic function closures while calculating bootstrap test statistics.

The code in `boot.null` has been amended to include the new choices of bootstrap-based null transformed test statistics described above. Originally, `boot.null` either returned the matrix of untransformed bootstrap test statistics or the matrix of centered and scaled null test statistics based on a logical value of the argument `csnull` (now deprecated, default was `TRUE`, indicating that centering and scaling should occur). The null shift and scale transformation was then done via the function `center.scale`, when `nulldist='boot'`. To accommodate the new diversity of bootstrap-based test statistics null distributions, two more null transformation functions have been written for the `boot.null` function. These functions are evaluated depending on the value of the character string passed to the `nulldist` argument in `MTP` or `EBMTP`. The three transforming functions are made available to the user upon loading the **multtest** library.

```
> args(center.scale)

function (muboot, theta0, tau0, alternative)
NULL

> args(center.only)

function (muboot, theta0, alternative)
NULL

> args(quant.trans)

function (muboot, marg.null, marg.par, ncp, alternative, perm.mat)
NULL
```

The function `center.scale` is evaluated if `nulldist='boot.cs'` or `'boot'` (corresponding to the old default value of the null distribution argument in the `MTP` function definition). The preferred new default value of `nulldist` is `'boot.cs'`, although code written for earlier releases of **multtest** will still compile using the now deprecated default `'boot'`. Similarly, `center.only` and `quant.trans` are evaluated if `nulldist='boot.ctr'` or `'boot.qt'`, respectively. Note that `center.only` is simply the `center.scale` function with the scaling argument `tau0` removed. Regardless of the choice of bootstrap null distribution, all null transformation functions are passed the character value of `alternative`, which ensures that sidedness is handled similarly as in the closures used for calculating observed test statistics.

Several function arguments in `MTP` and `EBMTP` have been changed or added to accommodate the new choices of bootstrap-based null distributions (Table 4). The most important new arguments for the null quantile-transformed null distribution are `marg.null` and `marg.par`. By default, these values are set to `NULL`, in which case the marginal null distribution used for the quantile transformation is selected based on choice of test statistic and possibly other parameters such as sample size $n$ (`ncol(X)`) or the number of unique class labels (`length(unique(Y))`, Table 5). In the case of Welch's $t$-statistics, i.e., two-sample $t$-statistics with unequal variance, the marginal null distribution is simply $\mathrm{N}(0,1)$, as determining a value for the effective degrees of freedom to apply over all hypotheses may be very context-specific. For tests of regression coefficients, $\mathrm{N}(0,1)$ was also selected as marginal null distribution. This decision was made to in order to reduce the number of arguments (object slots) which would have to be stored or passed to the `update` or `EBupdate` methods for obtaining results from several subsequent MTPs. (See details on the updating methods as well as the section on influence curve null distributions for more details.)

The choice of marginal null distribution when selecting `'boot.qt'` is somewhat more flexible than the null values used in the null shifted (and scaled) null distributions. The default values of `marg.null` and `marg.par` can be changed by the user should another null distribution be deemed more appropriate for the quantile-quantile transformation. One can specify the name of another parametric marginal null distribution by setting `marg.null` to one of `'normal'`, `'t'`, or `'F'`. Additionally, one can specify the parameters of the null distribution in `marg.par` in one of two ways, either by setting the values of the argument to a common numeric value, in which case the same value is applied to all hypotheses, or by passing `marg.par` a matrix with $M$ rows and columns equal to the number of parameters R would require to correctly define that distribution, e.g., `c(mean,sd)` for the normal distribution, or `c(df1,df2)` for an $F$-distribution. The following code gives example of exactly how to do this for Welchs's two-sample $t$-statistics allowing for unequal variance using the Chiaretti et al. [2004] dataset. Here, we use only the first 10 hypotheses, and restrict the number of bootstrap draws to `B=100`. Many more bootstrap samples are recommended in practice, particularly for larger numbers of hypotheses.

```
> qt.default <- MTP(X = X[1:10, ], Y = Ynum, nulldist = "boot.qt",
+     B = 100)

running bootstrap...
iteration = 100

> qt.tmarg <- MTP(X = X[1:10, ], Y = Ynum, nulldist = "boot.qt",
+     B = 100, marg.null = "t", marg.par = 30)

running bootstrap...
iteration = 100

> marg.pars <- matrix(c(rep(25, 5), rep(30, 5)), nr = 10, nc = 1)
> qt.tdifferent <- MTP(X = X[1:10, ], Y = Ynum, nulldist = "boot.qt",
+     B = 100, marg.null = "t", marg.par = marg.pars)

running bootstrap...
iteration = 100

> c(qt.default@marg.null, qt.tmarg@marg.null, qt.tdifferent@marg.null)

[1] "normal" "t"      "t"

> null.pars <- cbind(qt.default@marg.par, qt.tmarg@marg.par, qt.tdifferent@marg.par)
> colnames(null.pars) <- c("normal mean", "normal sd", "t df equal",
+     "t df vary")
> null.pars

      normal mean normal sd t df equal t df vary
 [1,]           0         1         30        25
 [2,]           0         1         30        25
 [3,]           0         1         30        25
 [4,]           0         1         30        25
 [5,]           0         1         30        25
 [6,]           0         1         30        30
 [7,]           0         1         30        30
 [8,]           0         1         30        30
 [9,]           0         1         30        30
[10,]           0         1         30        30
```

The above code chunk shows the results of three different calls to `MTP` with different settings of `marg.null` and `marg.par`. When `nulldist='boot.qt'`, objects of class *MTP* and *EBMTP* will return the values used by `marg.null` and `marg.par` in obtaining the estimate of the test statistics joint null distribution (Table 3). The values of the various `marg.null` and `marg.par` slots are shown in the code output above. The first call to `MTP` used the default marginal null distribution settings (Table 5), which, for two-sample Welch's $t$-statistics, correspond to the standard normal distribution. The first two columns of the `null.pars` matrix are the `mean` and `sd` parameters R uses to define the normal distribution. The last two calls to `MTP` demonstrate how the values of `marg.null` and `marg.par` can be manually set by the user.

Purely for illustrative purposes, the second call to `MTP` set the marginal null distribution to a $t$-distribution with $df = 30$. There may be applications, however, in which one may wish to vary the marginal null distribution over the family of hypotheses. In this way, one may account for missing observations or other factors relating to data quality when using a minP or *common-quantile* procedure. In this case, one may pass the `marg.par` argument a matrix with the desired values of the null distribution parameters for each hypothesis. In the previous example, the marginal null $t$-distribution was allowed to have $df = 25$ for the first five null hypotheses and then $df = 30$ for the last five null hypotheses. Both the `MTP` and `EBMTP` functions have built-in mechanisms ensuring that user-supplied values pertaining to the marginal null distribution make sense in terms of the value of `test`, and, in the case of `marg.par`, that the values also agree with choice of null distribution set in `marg.null`.

In terms of the specific implementation of the null quantile transformation method of van der Laan and Hubbard [2006], the function `quant.trans` obtains a sample of size $B$ from the specified parametric distribution. That sample is rearranged according to the row ranks of the untransformed bootstrap test statistics. One thing to consider, however, is that use of the quantile-transformed bootstrap null distribution will set the R random number generator ahead by $M \times B$ states.

Finally, as noted earlier, another choice of marginal null distribution may be derived from a data generating null distribution such as the permutation distribution. In cases where the marginal permutation distribution is known to be the exact marginal null distribution – e.g., for hypotheses of independence – the permutation distribution is a logical distribution to use for marginal quantile transformation. Unfortunately, for various reasons, the permutation null distribution implemented in **multtest** is not available for this use. One may, however, specify their own matrix of test statistics to use for quantile transformation by setting `marg.null='perm'` and passing a value to `perm.mat`. These values may be empirically derived elsewhere or, in certain circumstances, may be calculated based on theoretical information related to the testing problem at hand. For this purpose, the argument `perm.mat` can accept a matrix with rows equal to the number of hypotheses, each containing values from the test statistics distribution(s) to use for the qunatile transformation. When `marg.null='perm'`, the function `quant.trans` uses `approxfun` to approximate the marginal distribution functions. For this reason, the number of elements comprising the marginal null distribution does not have to equal the value set in `B`.

```
> set.seed(99)
> perms <- matrix(rnorm(5000), nr = 10, nc = 500)
> seed <- 926
> MTP.perm <- MTP(X[1:10, ], Y = Ynum, nulldist = "boot.qt", marg.null = "perm",
+     perm.mat = perms, B = 100, seed = seed)

running bootstrap...
iteration = 100

> MTP.perm@marg.null
```

```
[1] "perm"
```

In the toy example above, the bootstrap distribution was calculated for the first 10 hypotheses in the Chiaretti et al. [2004] data set using `B=100` bootstrap samples. A purely hypothetical matrix of test statistics, in this case a $10 \times 500$ matrix of independent normal test statistics, was used for the marginal quantile transformation. The value of the resulting `marg.null` slot is also shown. Conceptually, the matrix in `perm.mat` was originally intended to have a specific meaning – i.e., referring to the permutation distribution – which is why it was given a specific name. As as aside, passing values to `perm.mat` when `marg.null='perm'`, may also give the user more control of the R random number generator state in that specific (reproducible) realizations from a test statistics marginal null distribution may be used for quantile transformation rather than internally advancing the random number state $M \times B$ values (as is done when using the default settings for the `marg.null` and `marg.par` arguments in the `MTP` and `EBMTP` functions).

### 3.1.2 Influence curve null distributions

As already stated in the introduction, for a broad class of testing problems, such as the test of single-parameter null hypotheses using $t$-statistics, an appropriate, asymptotically valid null distribution is the $M$-variate Gaussian distribution $\mathrm{N}(0, \sigma^*)$, with mean vector zero and covariance matrix $\sigma^* = \Sigma^*(P)$ equal to the correlation matrix of the $M$-vector influence curve (IC) for an asymptotically linear estimator $\psi_n$ of the parameter of interest $\psi$ [Dudoit and van der Laan, 2008, Section 2.6]. Specifically, $\sigma^*$ may be estimated using the correlation matrix $\sigma_n^*$ corresponding to the $M \times M$ influence curve empirical covariance matrix,

$$\sigma_n = \hat{\Sigma}(P_n) = \frac{1}{n} \sum_{i=1}^{n} \mathsf{IC}_n(X_i) \mathsf{IC}_n^\top(X_i), \qquad (1)$$

where $\mathsf{IC}_n(X) = (\mathsf{IC}_n(X)(m) : m = 1, \ldots, M)$ is an estimator of the $M$-vector influence curve $\mathsf{IC}(X|P)$.

```
> args(corr.null)

function (X, W = NULL, Y = NULL, Z = NULL, test = "t.twosamp.unequalvar",
    alternative = "two-sided", use = "pairwise", B = 1000, MVN.method = "mvrnorm",
    penalty = 1e-06, ic.quant.trans = FALSE, marg.null = NULL,
    marg.par = NULL, perm.mat = NULL)
NULL
```

Null distributions based on influence curves have been derived for tests of means, linear model regression coefficients, and correlation coefficients [Gilbert et al., 2009, Dudoit and van der Laan, 2008, Section 2.6]. While bootstrap resampling is performed by the function `boot.null`, simulating from the multivariate normal distribution is carried out by the function `corr.null`. Both the `MTP` and `EBMTP` functions pass `corr.null` values specifying how the vectors of correlated mean-zero normal test statistics are to be generated. By default, null test statistics are obtained using the `mvrnorm` function in the `MASS` library [Venables and Ripley, 2002]. One may also use a Cholesky decomposition by setting the argument `MVN.method` to `'Cholesky'`. In this case, a small number (`penalty`) is added to the diagonal of the estimated test statistics correlation matrix in order to ensure the matrix is positive definite, preventing internal calls to `chol` from returning an error (Table 6).

Additionally, one may also apply the null quantile transformation of van der Laan and Hubbard [2006] to the multivariate normal test statistics joint null distribution obtained via influence curves. Because influence curve null distributions have only been implemented for $t$-statistics, the quantile transformation is only available for values of `marg.null` equal to `'t'` or `'perm'`. The quantile transformation is performed by a streamlined version of the `quant.trans` function called `tQuantTrans`. Default values for `marg.null` and `marg.par` apply as above, but, as before, may also be set manually by the user. In this specific setting, one important change has been made to the default marginal null distributions for regression coefficients in linear models, i.e., when `test='lm.XvsZ'` or `'lm.YvsXZ'` (Table 5). Because no matrix of raw bootstrap test statistics is generated when `nulldist='ic'`, it is not possible to subsequently `update` (`EBupdate`) the choice of null distribution for such objects of class *MTP* (*EBMTP*). This restriction placed on subsequent updating means that it is easier to set and retain values of `marg.null` and `marg.par` which may depend on other features of the data. Therefore, instead of using the standard normal distribution, the default marginal null distribution for linear regression parameters when `nulldist='ic'` and `ic.quant.trans=TRUE` becomes a $t$-distribution with $n - p$ degrees of freedom.

The next code chunk shows examples of the **multtest** interface when selecting `nulldist='ic'`. Again, we perform multiple testing in the toy example using the first ten genes from the Chiaretti et al. [2004] dataset. Due to the faster speed with which one can obtain an estimate of the test statistics null distibution using influence curves, we use the default setting `B=1000`.

```
> seed <- set.seed(926)
> MTP.ic.mvrnorm <- MTP(X[1:10, ], Y = Ynum, nulldist = "ic", seed = seed)

calculating vector influence curve...
sampling null test statistics...

> MTP.ic.chol <- MTP(X[1:10, ], Y = Ynum, nulldist = "ic", MVN.method = "Cholesky",
+     seed = seed)

calculating vector influence curve...
sampling null test statistics...

> MTP.ic.qt <- MTP(X[1:10, ], Y = Ynum, nulldist = "ic", ic.quant.trans = TRUE,
+     seed = seed)

calculating vector influence curve...
sampling null test statistics...
applying quantile transform...
```

The different options presented above correspond to the use of (i) `mvrnorm` or (ii) a Cholesky decomposition for obtaining vectors of correlated normal null test statistics. Option (iii) combines the use of `mvrnorm` with the $t$-distribution marginal quantile transformation of van der Laan and Hubbard [2006]. Adjusted $p$-values for this combination may be slightly higher, most likely owing to the use of a marginal $t$-distribution rather than relying on an asymptotic normal approximation.

Similar to the printed resampling output from the MTP and EBMTP functions, which display the number of completed permutation or bootstrap samples, when nulldist='ic', both functions will let the user know at what step each is at in the process of estimating the test statistics null distribution. If ic.quant.trans=TRUE, the function output will also let the user know if and when the marginal null quantile transformation is being carried out. Despite the computational efficiency often observed with the use of $t$-statistic-specific null distributions, it may still take several minutes to generate a large matrix of correlated null test statistics. In the example below, it appears there is little difference in the adjusted $p$-values when using the different options available with the influence curve-based null distribution.

```
> cbind(MTP.ic.mvrnorm@adjp, MTP.ic.chol@adjp, MTP.ic.qt@adjp)

           [,1]  [,2]  [,3]
100       0.742 0.756 0.750
10006     1.000 1.000 1.000
100132247 0.286 0.304 0.287
100132406 1.000 1.000 1.000
100133941 0.958 0.976 0.976
10014     1.000 1.000 1.000
10016     0.949 0.961 0.967
10019     1.000 1.000 1.000
10020     0.953 0.968 0.971
10036     0.847 0.872 0.880
```

As a last comment on the implementation of the influence curve null distributions for $t$-statistics, it should be noted that this choice of null distribution does support the use of weights, e.g., as specified in the argument W. In this case, regardless of choice of test statistic, the influence curve is calculated for each observation, rather than relying on calls to cor as is done for tests of means when W=NULL. The calculation of the corresponding null test statistics correlation matrix is achieved through the function IC.CorXW.NA, or, alternatively, IC.Cor.NA. These functions operate in a manner similar in spirit to the cov.wt function in R for computing weighted covariance matrices. One exception, however, is that both IC.CorXW.NA and IC.Cor.NA allow for the handling of NA elements in the vector influence curve in a way which eliminates their propagation throughout the matrices given by $\mathsf{IC}_n(X_i)\mathsf{IC}_n^\top(X_i)$. This is most similar to what would be done when specifying use='pairwise.complete.obs' in the cov or cor functions in R. This feature is crucial to the use of influence curve-based null distributions, as one cannot sample from a multivariate normal distribution using a covariance (correlation) matrix containing missing values.

## 3.2 Multiple testing procedures

### 3.2.1 Marginal adaptive FDR-controlling procedures

In the proof of its derivation, the original BH procedure required an independence assumption in order to guarantee Type I error control of the FDR [Benjamini and Hochberg, 1995]. In later work, Benjamini and Yekutieli [2001] proved the original procedure guarantees Type I error control under more relaxed conditions of *positive regression dependence*. When one is unsure these assumptions can be met, Benjamini and Yekutieli [2001] proposed a more conservative (essentially log-penalized) adaptation of the BH MTP. *In silico* experiments, however, have shown the BH procedure to perform well in a variety of conditions of practical interest in the genomics setting, e.g., when simulating multivariate normal data using empirical covariance matrices derived from microarray data. [Dudoit et al., 2008, Yoav Benjamini and Daniel Yekutieli, *personal communication*].

Adaptive linear step-up MTPs for controlling the FDR are 'adaptive' in the sense that they use the data to estimate the number (proportion) of true null hypotheses in an attempt to correct for the conservativeness of the original BH method. One gets more leverage out of adaptive MTPs, therefore, as the number true alternatives $h_1 = |\{\mathcal{H}_1\}|$ approaches the number of hypotheses $M$. The first of these adaptive Benjamini-Hochberg procedures [ABH; Benjamini and Hochberg, 2000] is based on graphical arguments regarding the behavior of raw $p$-values [Schweder and Spjøtvoll, 1982]. Another of these methods, a two-stage Benjamini-Hochberg procedure [TSBH; Benjamini et al., 2006], relies on a first-pass application of the original BH procedure. As a result, the corresponding adjusted $p$-values for this procedure become a function of the nominal significance threshold $\alpha$ used in the first stage [Dudoit et al., 2008]. In both cases the estimate is incorporated into the original BH procedure as a multiplicative correction factor. In the case of the TSBH procedure, note that if one uses a first-pass significance level of $\alpha = 0.10$, one will most likely want to control the FDR at the end of the second stage using a level of *at least* 0.10. In other words, when estimating the number of true null hypotheses $h_0$, it makes little sense in terms of guaranteeing Type I error control to assume a more liberal significance level at the first stage than the overall significance level at the end of the second stage.

```
> args(mt.rawp2adjp)

function (rawp, proc = c("Bonferroni", "Holm", "Hochberg", "SidakSS",
    "SidakSD", "BH", "BY", "ABH", "TSBH"), alpha = 0.05)
NULL
```

All marginal testing procedures in **multtest** are implemented in the function `mt.rawp2adjp`. The function accepts as first argument `rawp`, corresponding to a vector of raw (unadjusted) $p$-values for each hypothesis under consideration. These could be nominal $p$-values, for example, from $t$-tables or permutations. The next argument, `proc`, is a vector of character strings containing the names of the MTPs for which adjusted $p$-values are to be computed. This vector may include any of the following: `Bonferroni`, `Holm`, `Hochberg`, `SidakSS`, `SidakSD`, `BH`, `BY`, `ABH`, `TSBH`, corresponding to marginal MTPs for control of the FWER or FDR. The final argument `alpha` is a (vector of) nominal Type I error levels, used for estimating the number of true null hypotheses in the first stage of the two-stage Benjamini-Hochberg procedure (`TSBH`). The default value for `alpha` is `0.05`.

The function `mt.rawp2adjp` returns a list of values. In addition to a matrix of ordered, adjusted $p$-values `$adjp` and an index vector `$index` sorted according to the order of the original raw $p$-values, two additional list items are returned depending on whether or not either of the ABH or TSBH procedures were selected. If yes, the estimate $h_{0n}$ of the number of true null hypotheses is returned as a list element named `$h0.ABH` or `$h0.TSBH`. For the `TSBH` option, `mt.rawp2adjp` will check the length of the vector of the `alpha` argument and (i) insert the appropriate number of columns of adjusted $p$-values into the `$adjp` matrix and (ii) return a vector of the same length of `alpha` in `$h0.TSBH`. The vector of adjusted $p$-values is named with the nominal Type I error level pasted onto the element name (e.g., `h0.TSBH_0.05`).

In the toy example below, we switch from testing just 10 hypotheses as before, and we now focus on the first 100 filtered, gene-level expression measures in the Chiaretti et al. [2004] dataset. For use with the `mt.rawp2adjp` function, we retain the $M = 100$ raw $p$-values from the output of a call to MTP. As choice of null distribution, we used B=5000 vectors of null test statistics sampled from a mean-zero multivariate normal distribution with correlation matrix derived from the influence curves for differences of means. Using these empirical, raw $p$-values, we wish to control the both the FWER and FDR using the Bonferroni, BH, ABH, and TSBH MTPs. For the TSBH procedure, we will use first-pass nominal significance levels of both 0.05 and 0.10.

```
> seed <- set.seed(926)
> MTP.res <- MTP(X[1:100, ], Y = Ynum, nulldist = "ic", B = 5000,
+     seed = seed)

calculating vector influence curve...
sampling null test statistics...

> procs <- c("Bonferroni", "BH", "TSBH", "ABH")
> first.pass.alphas <- c(0.05, 0.1)
> adjp.out <- mt.rawp2adjp(MTP.res@rawp, proc = procs, alpha = first.pass.alphas)
> round(adjp.out$adjp[1:5, ], digits = 4)

       rawp Bonferroni     BH TSBH_0.05 TSBH_0.1    ABH
[1,] 0.0002       0.02 0.0100    0.0096   0.0091 0.0092
[2,] 0.0002       0.02 0.0100    0.0096   0.0091 0.0092
[3,] 0.0004       0.04 0.0133    0.0128   0.0121 0.0123
[4,] 0.0012       0.12 0.0300    0.0288   0.0273 0.0276
[5,] 0.0034       0.34 0.0571    0.0549   0.0520 0.0526

> adjp.out$index[1:5]

[1] 22 65 57 27 37

> adjp.out$h0.ABH

[1] 92

> adjp.out$h0.TSBH

h0.TSBH_0.05  h0.TSBH_0.1
          96           91
```

For space reasons, the above output displays only the adjusted $p$-value results and indices of the first five hypotheses (rather than the results of the full set of 100 hypotheses which were actually tested). One sees that the adjusted $p$-values for each of the adaptive Benjamini-Hochberg procedures are simply the results of the vanilla BH procedure multiplied the values of h0.ABH or h0.TSBH and then divided by the total number of hypotheses. For example, for the most significant result in the first row of the $adjp matrix, the Benjamini-Hochberg adjusted $p$-value of 0.0100 has been multiplied by the value of adjp.out$h0.ABH (92) and divided by the number of hypotheses (100), to give the adjusted $p$-value in the last column (0.0092) for the ABH procedure.

### 3.2.2 Empirical Bayes joint multiple testing procedures

The empirical Bayes multiple testing procedures (EBMTPs) have been formulated and characterized elsewhere [van der Laan et al., 2005, Dudoit et al., 2008, Dudoit and van der Laan, 2008, Chapter 7]. EBMTPs are intended to control generalized tail probability and expected value Type I error rates which can be defined as a parameter of the distribution of a function $g(V_n, S_n)$ of the numbers of false positives $V_n$ and true positives $S_n$ [Dudoit and van der Laan, 2008]. Examples of such Type I error rates include, among others, the FWER, gFWER, TPPFP and FDR. A central feature of the EBMTPs is the estimation of the distribution of *guessed sets of true null hypotheses* $Q_{0n}^{\mathcal{H}}$. A familiar choice of model for generating such guessed sets is the *common marginal non-parametric mixture model* [cf., among others, Efron et al., 2001, Storey, 2002, Storey and Tibshirani, 2003]. Here the $M$ test statistics are assumed to follow a *common marginal non-parametric mixture distribution*,

$$T_n(m) \sim f \equiv \pi_0 f_0 + (1-\pi_0) f_1, \qquad m = 1, \ldots, M, \tag{2}$$

where $\pi_0$ denotes the *prior probability of a true null hypothesis*, $f_0$ the *marginal null density of the test statistics*, and $f_1$ the *marginal alternative density of the test statistics*, i.e., $\pi_0 \equiv \Pr(H_0(m) = 1)$, $T_n(m)|\{H_0(m) = 1\} \sim f_0$, and $T_n(m)|\{H_0(m) = 0\} \sim f_1$.

Applying Bayes' rule to the elements comprising the test statistics mixture distribution in Equation (2) results in another parameter of interest, the *local q-value function*, i.e., the posterior probability function for a true null hypothesis $H_0(m)$, given the corresponding test statistic $T_n(m)$,

$$\pi_0(t) \equiv \Pr(H_0(m) = 1 | T_n(m) = t) = \frac{\pi_0 f_0(t)}{f(t)}, \qquad m = 1, \ldots, M. \tag{3}$$

In practice, the local $q$-value function $\pi_0(t)$ in Equation (3) is unknown, as it depends on the unknown true null hypothesis prior probability $\pi_0$, test statistic marginal null density $f_0$, and test statistic marginal density $f$. In many testing situations, the marginal null density is assumed to be known *a priori* and can be applied directly (e.g., N(0,1)). In the (re)sampling-based multiple testing case, an estimate $f_{0n}$ of the test statistics null distribution $f_0$ can be obtained by applying kernel density estimation over the pooled elements of the matrix comprising the null distribution estimate $Q_{0n}$. An estimate $f_n$ of the full density $f$ may also be obtained by applying density estimation over the vector of observed test statistics $T_n$ or, for a smoother estimate, the matrix of raw, untransformed bootstrap test statistics $\mathbf{T}_n^B$ (when available). Finally, as in Dudoit et al. [2008], a trivial estimator $\pi_{0n}$ of the prior probability $\pi_0$ of a true null hypothesis is the conservative value of one, i.e., $\pi_{0n} = 1$. Alternatively, under the assumption that the null hypotheses $H_0(m)$ are conditionally independent of the data $\mathcal{X}_n$ given the corresponding test statistics $T_n(m)$, the proportion of true null hypotheses $h_0/M \equiv \pi_0$ may be estimated less conservatively via the sum of the estimated local $q$-values,

$$\frac{h_{0n}^{QV}}{M} = \frac{1}{M} \sum_{m=1}^{M} \pi_{0n}(T_n(m)). \tag{4}$$

Having calculated the local $q$-value for each element in the vector of observed test statistics $T_n$, one can guess whether a given hypothesis is true by generating binary *Bernoulli realizations* of the corresponding posterior probabilities. Given a vector of null test statistics $T_{0n}$, a corresponding vector $H_{0n}$ of guessed true null hypotheses will partition $T_{0n}$ (similarly, $T_n$) into two sets of test statistics over which to count the numbers of guessed false positives $V_n$ (or the numbers of guessed true positives $S_n$) for a given cut-off $c$. In this way, a variety of Type I errors may be controlled [van der Laan et al., 2005, Dudoit and van der Laan, 2008, Dudoit et al., 2008, Gilbert et al., 2009].

Additionally, using the values of the observed test stastistics $T_n$ themselves as cut-offs, one may obtain multiple testing results in the form of adjusted $p$-values.

```
> args(EBMTP)

function (X, W = NULL, Y = NULL, Z = NULL, Z.incl = NULL, Z.test = NULL,
    na.rm = TRUE, test = "t.twosamp.unequalvar", robust = FALSE,
    standardize = TRUE, alternative = "two.sided", typeone = "fwer",
    method = "common.cutoff", k = 0, q = 0.1, alpha = 0.05, smooth.null = FALSE,
    nulldist = "boot.cs", B = 1000, psi0 = 0, marg.null = NULL,
    marg.par = NULL, ncp = NULL, perm.mat = NULL, ic.quant.trans = FALSE,
    MVN.method = "mvrnorm", penalty = 1e-06, prior = "conservative",
    bw = "nrd", kernel = "gaussian", seed = NULL, cluster = 1,
    type = NULL, dispatch = NULL, keep.nulldist = TRUE, keep.rawdist = FALSE,
    keep.falsepos = FALSE, keep.truepos = FALSE, keep.errormat = FALSE,
    keep.Hsets = FALSE, keep.margpar = TRUE, keep.index = FALSE,
    keep.label = FALSE)
NULL
```

The end-user interface of the EBMTP function shows much similarity to the function MTP. The same is also true when comparing slot values and names in the resulting objects of both the *EBMTP* and *MTP* classes. Because the estimation procedures inherent to both the vanilla MTPs and EBMTPs are so different, however, clear distinction between these classes was deemed both theoretically as well as practically important. Arguments to the function EBMTP which have been changed or added relative to MTP function arguments are listed and described in Table 7. Similarly, slots which have been changed or added in the output for objects of class *EBMTP* relative to objects of class *MTP* are listed and described in Table 8.

```
> args(Hsets)

function (Tn, nullmat, bw, kernel, prior, B, rawp)
NULL
```

Density estimation, local $q$-value estimation, and estimation of the distribution of guessed sets of true null hypotheses is performed by an internal call to the function Hsets. The Hsets function currently estimates both the test statistics null density $f_0$ and full density $f$ by applying kernel density estimation over the matrix of null test statistics and the vector of observed test statistics, respectively. Practically, more so than using the R functions dnorm or dt, for example, this step in EBMTP ensures that sidedness is correctly accounted for between the test statistics and their estimated null distribution. This step also allows EBMTPs to be applied to distributions of nonparametric test statistics (robust=TRUE) or non-standardized difference statistics (standardize=FALSE). Density estimation can be controlled using the arguments bw and kernel which are then used by the density function in R.

The prior probability $\pi_0$ of the local $q$-value function (Equation (3)) can be set to its most conservative value of 1 (prior='conservative') or estimated by some other means, e.g., using the adaptive Benjamini-Hochberg (prior='ABH') estimator (see also mt.rawp2adjp or the source code for the function ABH.h0) or by summing up the estimated local q-values themselves (prior='EBLQV') and then dividing by the number of total hypotheses $M$. Bounding these estimated probabilities by one produces a vector of estimated local $q$-values with length equal to the number of hypotheses. These estimated local $q$-values are returned by default in the slot `lqv`. Bernoulli (binary) realizations of the posterior probabilities indicate which hypotheses are guessed as belonging to the true set of null hypotheses given the value of their test statistics. The vectors of indicators are available in the slot `Hsets` when the argument `keep.Hsets` is `TRUE`.

Once this partitioning has been achieved, one can count the numbers of guessed false positives $V_n$ and guessed true positives $S_n$ which are obtained at each round of (re)sampling when using the value of an observed test statistic as a cut-off. The $M \times B$ matrix of guessed false (true) positives can be returned in the slot `falsepos` (`truepos`) when the argument `keep.falsepos` (`keep.truepos`) is `TRUE`. For computational speed, the counting of guessed false positives and guessed true positives is done using compiled `C` code stored in the source file `VScount.c`.

Note that the generation of Bernoulli realizations of the estimated local $q$-values relies on a call to `rbinom`. The user should be aware that this step will therefore set the `R` random number generator ahead $M \times B$ states.

```
> args(G.VS)

function (V, S = NULL, tp = TRUE, bound)
NULL
```

EBMTPs use closures generated by the function `G.VS` to represent Type I error rates in terms of their defining features. Restricting the choice of Type I error rate to those available in the package **multtest** – i.e., one of `typeone='fwer'`, `'gfwer'`, `'tppfp'`, or `'fdr'` – means that these features include (i) whether to control the number of false positives $V_n$ or the proportion of false positives among the number of rejections made $V_n/(V_n + S_n)$, (ii) whether one wishes to control a tail probability or expected value error rate, and, (iii) in the case of tail probability error rates, what bound should be placed on the random variable defining the Type I error rate (e.g., $k$ for the gFWER or $q$ for the TPPFP).

The function closures are then applied to the matrices of guessed false positives and guessed true positives, producing an $M \times B$ matrix with elements corresponding to guessed $g$-function-specific error rates for each hypothesis (test statistic) at each round of (re)sampling. This matrix may be returned in the `errormat` slot of an *EBMTP* object when the argument `keep.errormat` is `TRUE`. Averaging the Type I error results over $B$ (bootstrap or multivariate normal) samples provides an estimator of the evidence against the null hypothesis in the form of adjusted $p$-values obtained with respect to the choice of Type I error rate set in `typeone`. Finally, a monotonicity constraint is placed on the adjusted $p$-values before they are returned as output in the `adjp` slot.

As detailed in Dudoit et al. [2008], relaxing the prior $\pi_0$ of Equation (3) may result in a more powerful multiple testing procedure, albeit sometimes at the cost of Type I error control. Additionally, when the proportion of true null hypotheses is close to one, Type I error control may also become an issue, even when using the most conservative prior probability of one. The slot `EB.h0M` returned by objects of class *EBMTP* is the sum of the local $q$-values obtained via kernel density estimation divided by the total number of hypotheses $M$. If this value is close to one – heuristically $>$ than about 0.90 – the user will probably not want to relax the prior, as even the conservative EBMTP might be approaching a performance bound with respect to Type I error control. See website companion for Dudoit et al. [2008] for further details.

The user is advised to begin by using the most `'conservative'` prior, assessing the estimated proportion of true null hypotheses returned in the `EB.h0M` slot, and then deciding if relaxing the prior might be desired in that specific testing situation. New results with a different prior may be obtained by using the `EBupdate` method. Gains in power over other MTPs, however, have been observed even when using the most conservative prior of one. Situations of moderate-high to high levels of correlation may also affect the Type I error performance of multiple testing methods which use the same non-parametric mixture model for generating $q$-values [cf., Storey, 2002]. Microarray analysis represents a scenario in which dependence structures are typically 'weak enough' to mitigate this concern. On the other hand, the analysis of densely sampled SNPs, for example, may present difficulties.

Using the dataset of Chiaretti et al. [2004], we again compute two-sample Welch's $t$-statistics to probe null hypotheses corresponding to no differential expression between the two subgroups of ALL patients. Here we restrict ourselves to the first 100 genes in the dataset, and use the test statistics null distribution obtained from the vector influence curves. The specific case of FDR control when using the default conservative prior and `B=2500` samples of null test statistics is shown.

```
> ebFdrIc <- EBMTP(X = X[1:100, ], Y = Ynum, nulldist = "ic", typeone = "fdr",
+     B = 2500)

calculating vector influence curve...
sampling null test statistics...

counting guessed false positives...
250 500 750 1000 1250 1500 1750 2000 2250 2500

counting guessed true positives...
250 500 750 1000 1250 1500 1750 2000 2250 2500
```

Note that the printed output of `EBMTP` also lets the user know where the function `VScount` is in the process of counting the guessed numbers of true positives and false positives. The next chunk shows the similarity between the output of the `summary` method when operating on an object of *EBMTP* when compared to operating on an object of class *MTP*. As additional information, the type of prior used in the EBMTP is also given in the summary output.

```
> summary(ebFdrIc)

EBMTP: common.cutoff
Type I error rate:  fdr
prior:  conservative

  Level Rejections
1  0.05          9

                Min.  1st Qu.  Median    Mean 3rd Qu.   Max. NA's
adjp         0.01237  0.34770 0.56670 0.49860  0.7146 0.8063    0
rawp         0.00000  0.11830 0.32600 0.36700  0.6176 1.0000    0
statistic -3.09900 -0.70610 0.37840 0.32930  1.2500 3.8220    0
estimate  -0.31170 -0.09439 0.05589 0.05809  0.1664 0.8236    0
```

```
> par(mfrow = c(2, 2))
> plot(ebFdrIc)
```

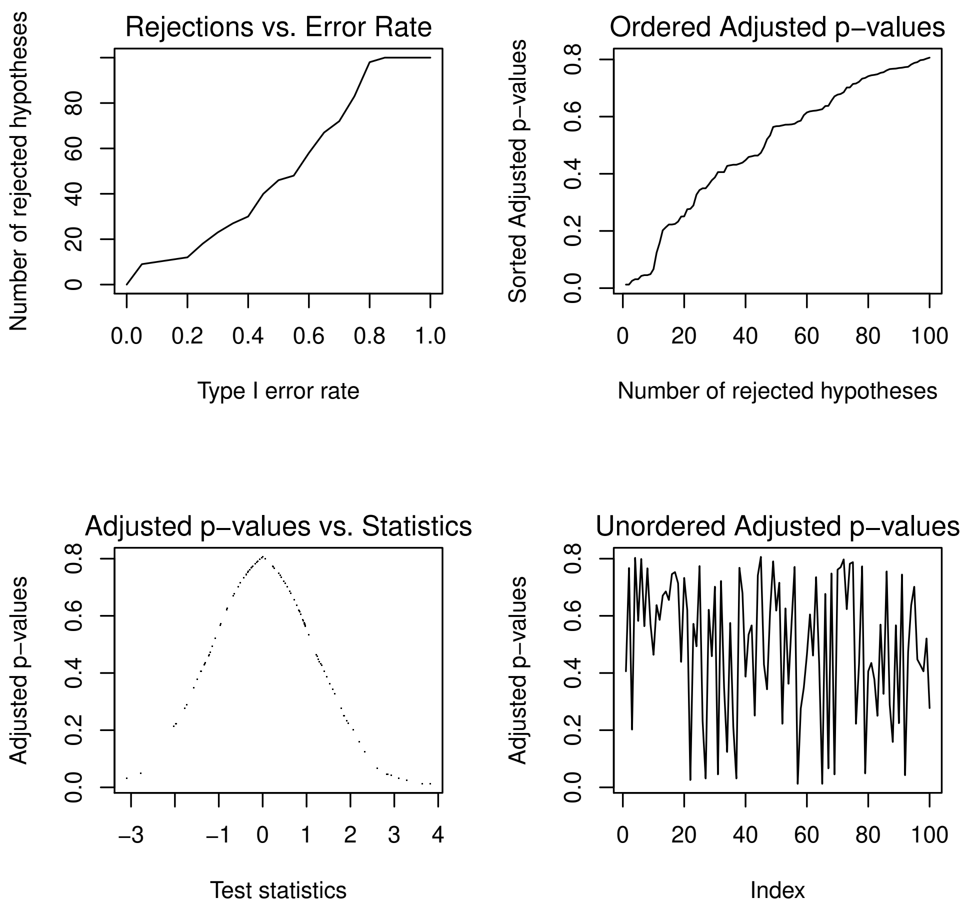

Figure 1: *Default output from the* `plot` *method for objects of EBMTP in the* ***multtest*** *package.*

Finally, four summary and diagnostic plots are available through the `plot` method for objects of class *EBMTP* (Figure 1). These figures allow one to track the behavior of the relationship between the Type I error rate with the number of rejected hypotheses (Figure 1, top) as well as more closely explore properties of the estimated adjusted $p$-values returned by the call to `EBMTP` (Figure 1, bottom). Some additional plots available for objects of class *MTP* are not available for the EBMTPs, because they rely on the calculation of test statistic cut-offs or confidence regions for each hypothesis, neither one of which are currently available from the function `EBMTP`.

## 4 Application: Acute lymphoblastic leukemia microarray data

### 4.1 Data and preprocessing

We illustrate the additional features of the **multtest** package using the microarray dataset of Chiaretti et al. [2004]. Taken from a gene expression study of patients with acute lymphoblastic leukemia (ALL), the data are publicly available in the **ALL** experimental data package on the Bioconductor website (`http://www.bioconductor.org`).

The main object in this package is `ALL`, an instance of the class *ExpressionSet*, which contains the mRNA expression measures, patient phenotypes, and gene annotation information. The genes-by-subjects matrix of expression measures is provided in the `exprs` slot of `ALL` and the phenotype data are stored in the `phenoData` slot. Note that the expression measures have been obtained using the three-step robust multichip average (RMA) pre-processing method, implemented in the package **affy** [Gautier et al., 2004]. In particular, the expression measures have been subject to a base 2 logarithmic transformation.

```
> library(ALL)
> data(ALL)
> slotNames(ALL)

[1] "assayData"         "phenoData"         "featureData"
[4] "experimentData"    "annotation"        ".__classVersion__"
```

While the original goal of the analysis in Chiaretti et al. [2004] was the identification of distinct subsets of patients with ALL, we will use the dataset to explore differential gene expression between two groups of affected individuals. In particular, we are interested in assessing differential expression between patients positive and negative for the BCR/Abl fusion phenotype. The BCR/Abl fusion, also known as the Philadelphia chromosome, is a translocation of regions on human chromosomes 9 and 22. Patients positive for this phenotype therefore represent a cytogenetically distinct subset of ALL affected individuals. There are a total of 79 patients with these particular phenotype values.

```
> varLabels(ALL)

 [1] "cod"             "diagnosis"      "sex"            "age"
 [5] "BT"              "remission"      "CR"             "date.cr"
 [9] "t(4;11)"         "t(9;22)"        "cyto.normal"    "citog"
[13] "mol.biol"        "fusion protein" "mdr"            "kinet"
[17] "ccr"             "relapse"        "transplant"     "f.u"
[21] "date last seen"

> BT <- sapply(ALL$BT, substring, 1, 1) == "B"
> mol.biol <- (ALL$mol.biol == "BCR/ABL") | (ALL$mol.biol == "NEG")
> whichPheno <- mol.biol & BT
> n <- sum(whichPheno)
> Y <- as.vector(ALL$mol.biol[whichPheno])
> table(Y)
```

```
Y
BCR/ABL     NEG
     37      42
```

```
> Ynum <- rep(1, 79)
> Ynum[Y == "NEG"] <- 0
```

We employ probe filtering as in von Heydebreck et al. [2004]. In particular, we wish to retain gene expression measures with absolute intensities $\geq 100$ for at least 25% of the patient samples. Additionally, probes were selected to have an interquartile range of at least 0.5 on the $\log_2$ scale. Finally, using the annotation database package **hgu95av2.db** (`http://www.bioconductor.org`), probe intensities mapping to the same gene were averaged, resulting in a final dataset consisting of 2051 gene-level expression measures on 79 patients.

```
> library(hgu95av2.db)
> whichGeno <- apply(exprs(ALL[, whichPheno]), 1, function(z) ((mean(2^z >
+     100) >= 0.25) & (IQR(z) > 0.5)))
> subALL <- ALL[whichGeno, whichPheno]
> AffyID <- featureNames(subALL)
> EntrezID <- unlist(mget(AffyID, env = hgu95av2ENTREZID))
> X <- aggregate(exprs(subALL), list(EntrezID = factor(EntrezID)),
+     mean)
> rownames(X) <- X[, "EntrezID"]
> X <- X[, -1]
> colnames(X) <- Y
> dim(X)
```

```
[1] 2051   79
```

## 4.2 Data analysis and computational efficiency

The goal of this section is to illustrate what types of results and performance to expect when using the `MTP` and `EBMTP` functions in more realistic settings. Using the dataset of Chiaretti et al. [2004], sample code snippets will be displayed. Most of the results below have been obtained from running a separate code file which is available in the supplementary material of this paper. In addition to presenting the results of the individual MTPs that were applied to the data, we will also present the run times associated with each procedure. Finally, a comparison of the test statistics null distribution estimates using the bootstrap-based approach and the method using influence curves will also be presented.

### 4.2.1 Differential gene expression

As before, we will begin by testing for differential gene expression among patients with ALL, comparing patients with the BCR/Abl translocation to patients with cytogenetically normal disease. The code below illustrates how one might implement the single-step maxT procedure (ss maxT) using two-sample Welch's $t$-statistics allowing to allow for unequal variance between groups. Both of these selections are defaults in the MTP function definition. We will begin by using the null quantile-transformed bootstrap test statistics null distribution of van der Laan and Hubbard [2006], with default $\mathrm{N}(0,1)$ marginal null distributions and B=5000 bootstrap samples.

Because we are also partially interested in comparing the results of several MTPs using different choices of test statistics null distributions, the keep.rawdist argument has been set to TRUE. This allows for new bootstrap-based null distributions to be estimated in future calls to the update method using the untransformed matrix of bootstrap test statistics from the original *MTP* object (e.g., in our case, via the other internal null transformation functions center.scale, center.only). In practice, both of the matrices stored in the rawdist and nulldist slots of an *(EB)MTP* object can become quite large. Therefore, the update (similarly, EBupdate) method only requires that one of these matrices are stored in order to perform additional MTPs. If one wishes to compare different choices of bootstrap-based null distribution, then the untransformed bootstrap test statistics must be saved in the rawdist slot in order for the new null transformations to be calculated. In the case of an empty nulldist slot, the null distribution will be recalculated from the matrix in the rawdist slot during the next call to update. If one knows in advance that they will not wish to compare different bootstrap null distribution estimates, then using the default keep.nulldist=TRUE setting in the original function call is both necessary and sufficient for subsequent updating.

```
> seed <- set.seed(926)
> MTP.qt <- MTP(X, Y = Ynum, nulldist = "boot.qt", keep.rawdist = TRUE,
+     B = 5000, seed = seed)
> MTP.cs <- update(MTP.qt, nulldist = "boot.cs")
> MTP.ctr <- update(MTP.qt, nulldist = "boot.ctr")
> MTP.ic <- MTP(X, Y = Ynum, nulldist = "ic", B = 5000, seed = seed)
```

Because the influence curve null distribution is a stand-alone null distribution which directly calculates a matrix of null test statistics, one cannot update the null distribution argument from the MTP.qt object of class *MTP* in the same way as for the other bootstrap null distributions. It is better in this case to simply use a separate call to MTP. One may, however, update other MTP features from an object of class *MTP* or *EBMTP* which initially used the influence curve approach for estimating the test statistics null distribution, e.g., method, alternative, typeone, etc.

The next two commands show how one might use a call to update in order to obtain augmentation multiple testing procedure (AMTP) results from the original object MTP.qt for control of the gFWER or TPPFP. Again, AMTPs use the rejection (adjusted $p$-value) results of the initial FWER-controlling procedure in order to produce the results for gFWER, TPPFP or FDR control. The specific cases of gFWER($k$=5) and TPPFP($q$=0.10) are shown, where $k$ is the number of allowed false positive and $q$ represents a bound on the proportion of false positives among the rejection made.

```
> AMTP.gfwer5 <- update(MTP.qt, typeone = "gfwer", k = 5)
> AMTP.tppfp10 <- update(MTP.qt, typeone = "tppfp", q = 0.1)
```

The EBMTP command functions similarly to the function MTP. What follows are examples of calls to EBMTP which perform the common cut-off EBMTPs using the default conservative prior. Both the cases showing the bootstrap-based null quantile-transformed and influence curve-derived test statistics joint null distributions are presented.

```
> EB.fwer.qt <- EBMTP(X, Y = Ynum, nulldist = "boot.qt", B = 5000,
+     seed = seed)
> EB.fwer.ic <- EBMTP(X, Y = Ynum, nulldist = "ic", B = 5000, seed = seed)
```

To save on computation time, it is possible to use the resampling results from an object of class *MTP* in order to more quickly execute EBMTPs. The opposite is also true. This is achieved through the class conversion methods `mtp2ebmtp` and `ebmtp2mtp`. In the code below, once the new object of class *EBMTP* has been created, EBMTP results can be obtained via calls to the `EBupdate` method. With the exception of the FDR-controlling EBMTPs performed below, unless stated otherwise in the function calls, the default prior probability of one, i.e., `prior=‘conservative’` was used. The value returned in the `EB.h0M` slots of the various *EBMTP* objects was around 0.78 (data not shown), indicating that one may possibly want to relax the choice of prior.

```
> newEB <- mtp2ebmtp(MTP.qt)
> EB.fwer <- EBupdate(newEB)
> EB.gfwer5 <- EBupdate(EB.fwer, typeone = "gfwer", k = 5)
> EB.tppfp10 <- EBupdate(EB.fwer, typeone = "tppfp", q = 0.1)
> EB.fdr <- EBupdate(EB.fwer, typeone = "fdr")
> EB.fdr.abh <- EBupdate(EB.fwer, typeone = "fdr", prior = "ABH")
> EB.fdr.lqv <- EBupdate(EB.fwer, typeone = "fdr", prior = "EBLQV")
```

Recall that the EBMTPs generate multiple testing results for non-FWER error rates in a very different way than the AMTPs did above. Due to the number of arguments which may affect the results of an EBMTP, particularly when generating vectors of guessed null hypotheses in the `Hsets` function, i.e., `nulldist`, `prior`, `bw`, `kernel`, etc., the counting of guessed false positives and guessed true positives (again, done in `C` by the function `VScount`) is currently automatically repeated for each call to `EBupdate`.

Additionally, when calling `EBupdate` for any Type I error rate other than FWER, the `typeone` argument must be specified (even if the original object did not control FWER). For example, `typeone="fdr"`, would always have to be specified, even if the original object also controlled the FDR. In other words, for all function arguments, it is best to always assume that you are updating from the `EBMTP` default function settings, regardless of the original call to the `EBMTP` function. Presently, the main advantage of the `EBupdate` method is that it prevents the need for repeated estimation of the test statistics null distribution.

Using the dataset of Chiaretti et al. [2004], one sees that the run times associated with null distributions obtained from the vector influence curves are considerably faster than those of the null distributions estimated by transformed bootstrap test statistics (Table 9). The use of the methods `update` and `EBupdate`, by reducing the need for repeated estimation of the test statistics joint null distribution, has also drastically reduced the amount of required computation time to produce results for different MTPs and for different choices of Type I error rates (Table 9). The longer run times associated with the calls to `EBupdate` compared to `update` reflect the complexity of EBMTPs relative to AMTPs and are the byproduct of recounting the numbers of false positives and (when applicable) true positives, Table 9).

In the case of FWER control, the use of the influence curve null distribution and the empirical Bayes methods resulted in the most rejections (Table 9). For control of the gFWER and TPPFP, the empirical Bayes methods rejected many more hypotheses than the AMTPs. In the case of gFWER(5) control, one can see that the AMTP has simply rejected $k = 5$ additional hypotheses

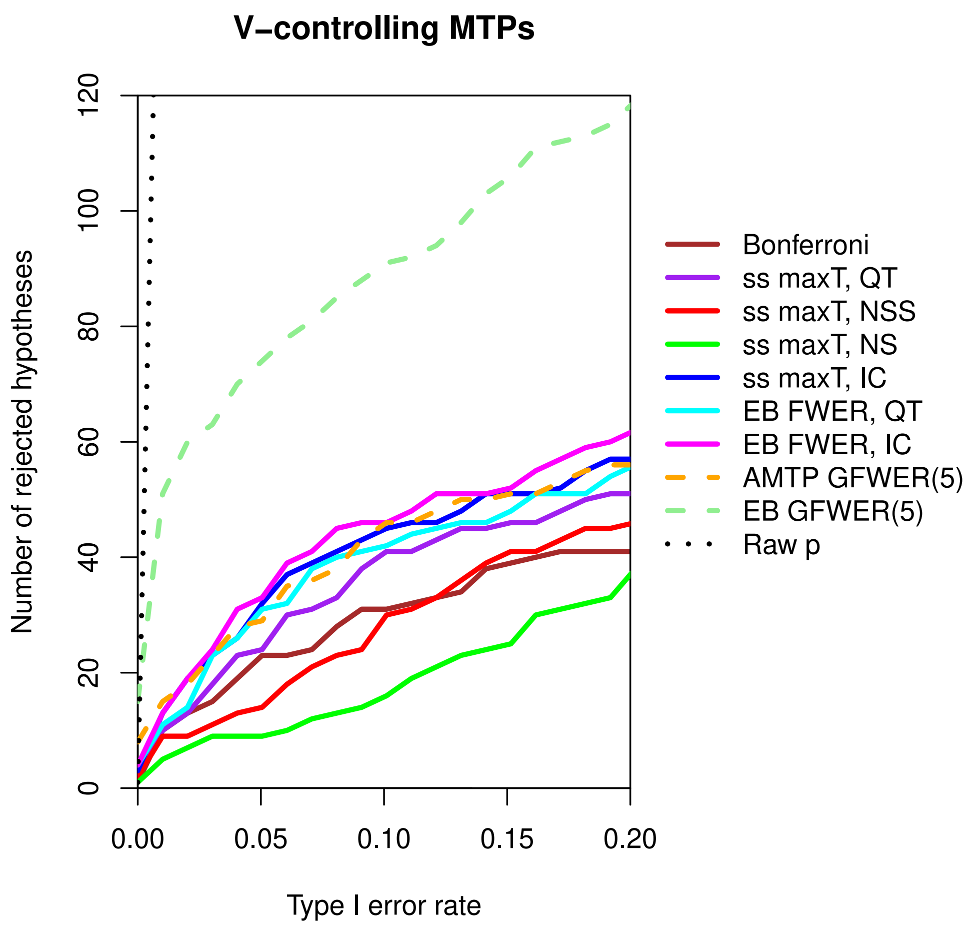

Figure 2: *Graphical summary for select MTPs controlling the tail probability of the number of false positives $V_n$ using the dataset of Chiaretti et al. [2004].* This figure was generated using the `mt.plot` function in the **multtest** package. EBMTPs perform competitively with other MTPs as evidenced by the number of rejected hypotheses observed over a range of significance levels.

relative to the `ss maxT` procedure using the null quantile transformation (Table 9, rows 2 and 9). EBMTPs, therefore, represent a potentially useful tool for characterizing the behavior of less-understood, more general Type I error rates such as the gFWER or the TPPFP. Finally, the FDR-controlling EBMTPs consistently reject more hypotheses than even the marginal adaptive Benjamini-Hochberg procedures (Table 9). Graphical summaries of the MTP results obtained using the function `mt.plot` are shown in Figures 2 and 3.

### 4.2.2 Comparison of test statistics null distributions

We were also interested in comparing the estimates of the test statistics joint null distributions using both the boostrap-based and influence curve-based methods. Given the significant decrease in time associated with sampling from a multivariate normal distribution as compared to performing bootstrap calculations, one might be interested if this improved computational efficiency is achieved

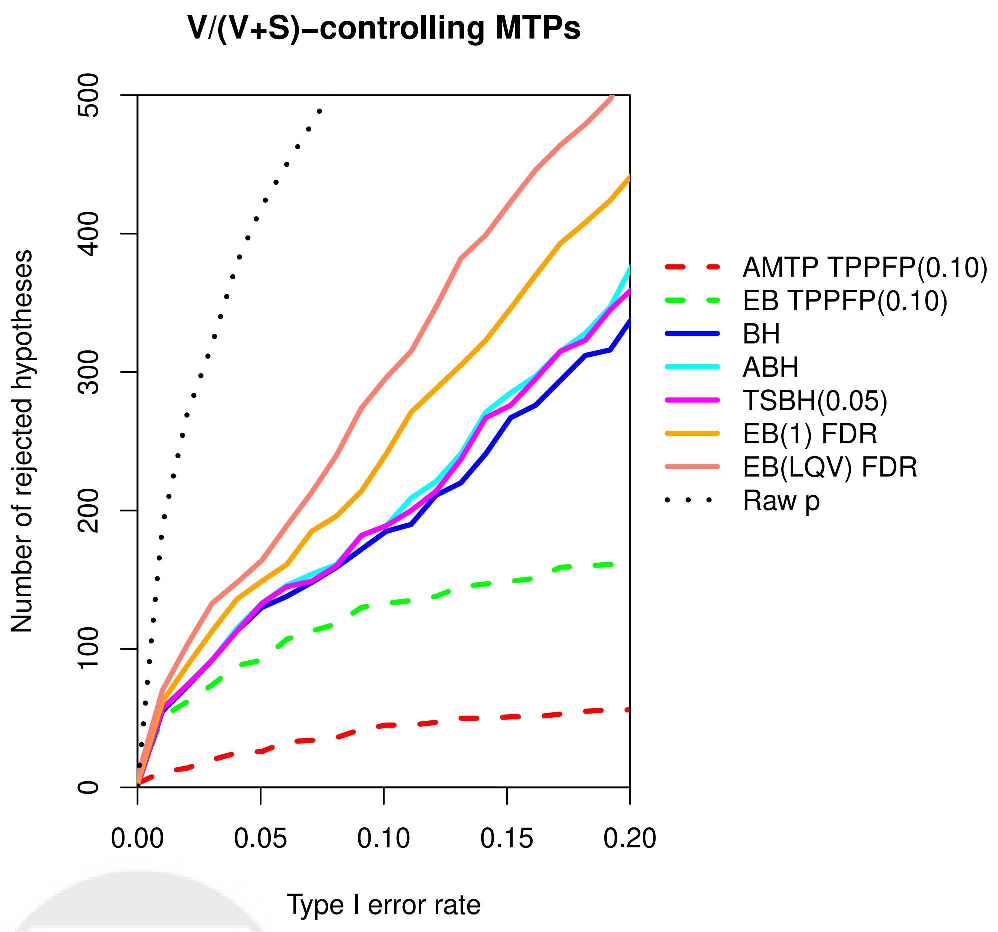

Figure 3: *Graphical summaries for select MTPs controlling the false discovery proportion,* $V_n/(V_n + S_n)$. This figure was generated using the `mt.plot` function in the **multtest** package. EBMTPs perform competitively with other MTPs as evidenced by the number of rejected hypotheses observed over a range of significance levels.

at the cost of noticeable differences between the two approaches.

To explore this question, we decided to compare unique elements of the empirical genes $\times$ genes correlation matrix obtained from the vectors of null test statistics in two settings: (i) two-sample tests of means, and (ii) regression coefficients in linear models. The null test statistics were obtained using the null-shifted test statistics null distribution (center-transformed, `nulldist='boot.ctr'`) and the influence curve null distribution (`nulldist='ic'`).

The first comparison for two-sample tests of means proceeds as before. Use of the argument `alternative='greater'` ensures that the null test statistics returned in `nulldist` slot have not undergone an absolute value transformation (done internally by `MTP` and `EBMTP` to handle sidedness for two-sided null hypotheses).

```
> seed <- set.seed(926)
> MTP.ctrg <- MTP(X, Y = Ynum, nulldist = "boot.ctr", alternative = "greater",
+     B = 5000, seed = seed)
> MTP.icg <- MTP(X, Y = Ynum, nulldist = "ic", alternative = "greater",
+     B = 5000, seed = seed)
```

The second comparison between the two estimates of the test statistics null distributions involves tests of regression parameters in linear models. Here, gene expression for the $m$th variable is the outcome, while the regression coefficient on a variable representing the presence of the BCR/Abl phenotype is our parameter of interest in a linear model which adjusts for the gender and age of the patient, i.e.,

$$\mathsf{E}[X(m)|Z_1, Z_2, Z_3] = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \beta_3 Z_3, \tag{5}$$

where $X(m)$ is the gene expression measurements for the $m$th hypothesis (gene), $Z_1$ is an indicator variable equal to one if the patient has the BCR/Abl fusion, $Z_2$ is an indicator variable for the gender of the patient, and $Z_3$ represents the patient's age. We are interested in testing the two-sided null hypotheses $H_0(m) = \mathsf{I}(\beta_1(m) = 0)$ vs. $H_1 = \mathsf{I}(\beta_1(m) \neq 0)$.

A gene expression matrix `X.lm` and a design matrix `covars` were created in which patients with missing covariate data have been removed prior to the analysis, leaving $n = 76$ patients total. The design matrix needs not contain a column vector for an intercept term, as an intercept is automatically assumed by the **multtest** linear model test statistic function closures.

```
> covars <- cbind(Ynum, ALL$sex[whichPheno], ALL$age[whichPheno])
> colnames(covars) <- c("bcrAbl", "sex", "age")
> rms <- apply(covars, 1, function(z) sum(is.na(z)))
> rm.these <- which(rms != 0)
> covars <- covars[-rm.these, ]
> X.lm <- X[, -rm.these]
> dim(X.lm)
```

Calls to `MTP` are then made by specifying the values of `test` and the variables of the design matrix to be used in the linear model.

```
> seed <- set.seed(926)
> MTPlm.ctrg <- MTP(X.lm, Z = covars, Z.test = 1, Z.incl = c(2,
+     3), test = "lm.XvsZ", nulldist = "boot.ctr", alternative = "greater",
+     B = 5000, seed = seed)
> MTPlm.icg <- MTP(X.lm, Z = covars, Z.test = 1, Z.incl = c(2,
+     3), test = "lm.XvsZ", nulldist = "ic", alternative = "greater",
+     B = 5000, seed = seed)
```

For each matrix of null test statistics returned in a given object's `nulldist` slot, the genes $\times$ genes correlation matrix was obtained using the `cor` command. For each choice of test statistic, corresponding unique elements of the correlation matrices obtained from the matrix of null-centered bootstrap test statistics and from the matrix of correlated normal null test statistics were compared (Figure 4). Figures were generated using the **geneplotter** [Gentleman and Biocore, 2009] package with colors from **RColorBrewer** [Neuwirth, 2007]. By examining the densities of the differences between like elements of the estimated null test statistics correlation matrices, one observes these two approaches to yield relatively similar results in terms of their ability to estimate the dependencies between (null) test statistics. The differences between corresponding elements of the null test statistics correlation matrices appear to be mean-centered, and they also do not appear to be affected by the strength of correlation (Figure 4). Since both approaches rely on asymptotics, one would expect the results presented in Figure 4 to also be a function of the sample size ($n$), the dimensionality of the testing problem ($M$), and the number of samples of null test statistics ($B$) used in estimating the test statistics null distribution.

Again, a striking observation is the decrease in computational time associated with influence curve-based null distrbutions. Similar to the results in Table 9, for two-sample Welch's $t$-statistics, the bootstap method required 1h 16m 31s of system time, while setting `nulldist='ic'` returned a result in 3m 41s. This difference was even more noteworthy when testing regression parameters (which take longer to calculate in **multtest** than simpler two-sample test statistics). Here the bootstrap method required 1h 54m 30s of system time as compared to just 3m 46s when using influence curves derived for regression parameters. All code was executed on a 64-bit Linux machine with 4GB RAM running R-2.8.0.

```
> sessionInfo()

R version 2.8.0 (2008-10-20)
x86_64-redhat-linux-gnu

locale:
LC_CTYPE=en_US.iso885915;LC_NUMERIC=C;LC_TIME=en_US.iso885915;
LC_COLLATE=en_US.iso885915;LC_MONETARY=C;LC_MESSAGES=en_US.iso885915;
LC_PAPER=en_US.iso885915;LC_NAME=C;LC_ADDRESS=C;LC_TELEPHONE=C;
LC_MEASUREMENT=en_US.iso885915;LC_IDENTIFICATION=C

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] hgu95av2.db_2.2.5  RSQLite_0.6-7      DBI_0.2-4
[4] AnnotationDbi_1.4.3 ALL_1.4.4          multtest_1.23.6
[7] Biobase_2.2.2
```

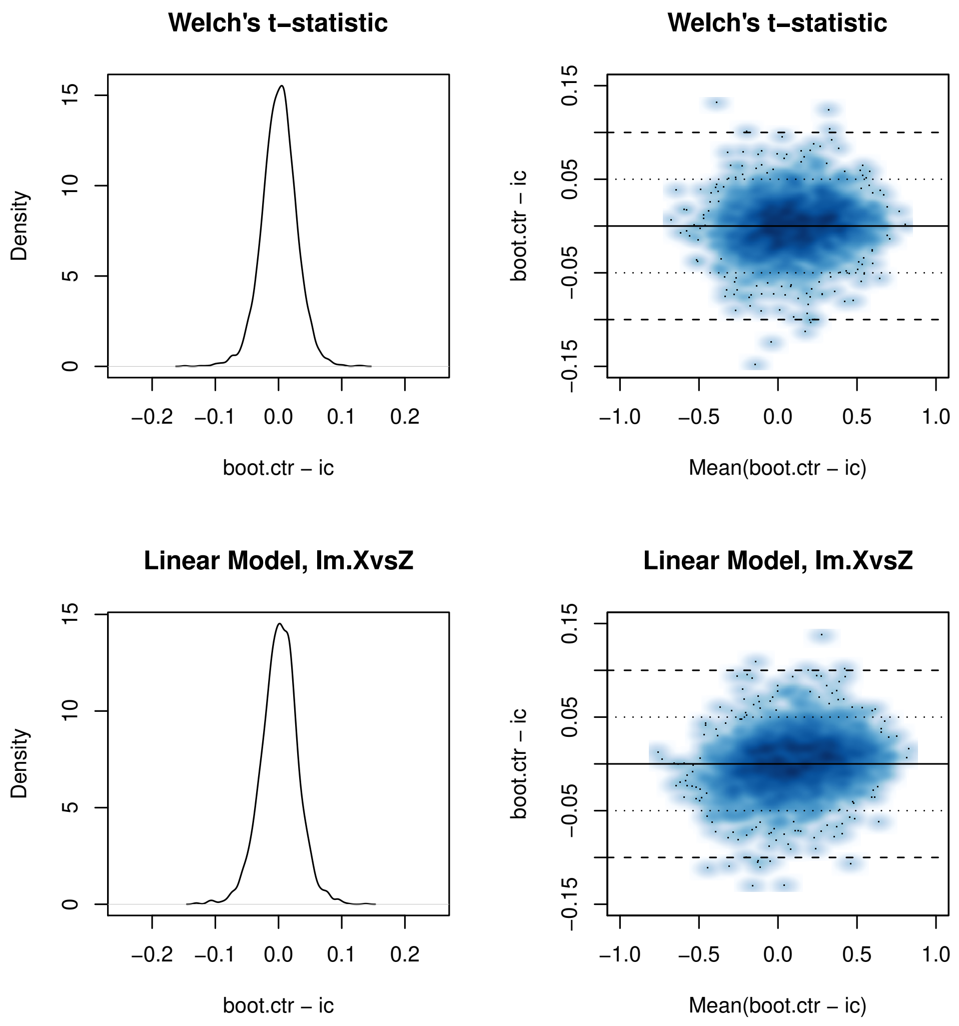

Figure 4: *Comparison of 2500 unique elements of the test statistics null distribution empirical correlation matrices as estimated by the null shifted test statistics null distribution (*`nulldist='boot.ctr'`*) and the null distribution obtained through influence curves (*`nulldist='ic'`*).* Comparisons are shown for two-sample $t$-statistics with unequal variance (`test='t.twosamp.unequalvar'`), i.e., Welch's $t$-statistics, comparing gene expression among ALL patients with and without the BCR/Abl fusion (top), and (ii) for the coefficient on the BCR/Abl fusion term in a linear model with gene expression as outcome, adjusting for age and gender of the patients using `test='lm.XvsZ'` (bottom). For each estimated test statistics null distribution returned in the genes $\times B$ `nulldist` slot of the respective *MTP* objects, 2500 unique elements of the genes $\times$ genes correlation matrix were obtained. Plots of the density of the difference between corresponding elements of each estimated correlation matrix (left) as well as smooth scatter plots showing the differences as a function of mean correlation between the elements (right) are displayed. Superimposed points on the density cloud images represent the top 5% outliers. The differences appear to be closely centered around zero and do not seem to vary considerably with the strength of the correlation being estimated.

```
loaded via a namespace (and not attached):
[1] MASS_7.2-40     splines_2.8.0   survival_2.34-1
```

## 5 Conclusion

The added functionality of the **multtest** package should aid the practitioner in being able to study as well as perform a wide variety of joint MTPs. With an expanded repertoire of test statistics null distributions and the inclusion of EBMTPs, powerful alternatives to traditional approaches to multiple testing are now available for use. The inclusion of the test statistics null distribution estimate based on samples of mean-zero correlated null test statistics is also very appealing, as it alleviates potential work flow bottlenecks associated with resampling-based joint MTPs. Future work may focus on reducing redundant computation time associated with the method `EBupdate` as well as implementing formula-based options for handling input data objects (e.g., as in the function `lm`).

| Object | Description |
|---|---|
| `X` | A matrix, data frame or *ExpressionSet* containing the raw data. In the case of an *ExpressionSet*, `exprs(X)` is the data of interest while `pData(X)` may contain outcomes and covariates of interest. Variables are arranged in rows; observations are arranged in columns. |
| `W` | A vector or matrix containing non-negative weights to be used in computing the test statistics. Weights can vary by samples, variables, or both. Weights received as vectors are duplicated appropriately to produce a matrix `W` with the same dimension as `X` with one weight for each value in `X`. |
| `Y` | A vector, factor, or `Surv` object containing the outcome of interest. Depending on choice of test, `Y` may be a vector of class labels, a dependent variable in a linear model or survival data. |
| `Z` | A vector, factor, or matrix containing covariate data to be used in the regression models. Each variable should be in one column, such that `nrow(Z)=ncol(X)` |
| `Z.incl` | The indices or names of columns of `Z` designating which variables to include in a regression model. |
| `Z.test` | The index or name of the column of `Z` to use to test for association with each row of `X` in a linear model. Only used for `test='lm.XvsZ'`, where it is necessary to specify which covariate's regression parameter is of interest. |
| `na.rm` | Logical indicating whether to remove `NA` observations. Default is `'TRUE'`. |

Table 1: Data objects supported by the **multtest** package functions `MTP` and `EBMTP`.

| Test Statistic | Description | Null Distribution |
|---|---|---|
| `t.onesamp` | One-sample $t$-statistic for tests of means. | `boot,ic` |
| `t.twosamp.-equalvar` | Two-sample $t$-statistic with equal variance for difference in means. | `perm,boot,ic` |
| `t.twosamp.-unequalvar` | Two-sample $t$-statistic with unequal variance for difference in means, i.e., two-sample Welch's $t$-statistic. | `perm,boot,ic` |
| `t.pair` | Two-sample paired $t$-statistic for tests of differences in means. | `perm,boot,ic` |
| `f` | Multi-sample $F$-statistic for tests of equality of population means. | `perm,boot`$^\dagger$ |
| `f.block` | Multi-sample $F$-statistic for tests of equality of population means in a block design. | `perm,boot`$^\dagger$ |
| `f.twoway` | Multi-sample $F$-statistic for tests of equality of population means in a block design. Differs from `f.block` in requiring multiple observations per group-block combination. See help files for details. | `boot`$^\dagger$ |
| `lm.XvsZ` | Regression coefficient $t$-statistic for variable `Z.test` in linear models, possibly adjusted by covariates `Z.incl` from the matrix `Z`. In the case of no covariates, one recovers the one-sample $t$-statistic, `t.onesamp`. | `boot,ic` |
| `lm.YvsXZ` | Regression coefficient $t$-statistic in linear models, with outcome `Y` and each row of `X` as covariate of interest, with possibly other covariates `Z.incl` from the matrix `Z`. | `boot,ic` |
| `coxph.YvsXZ` | Regression coefficient $t$-statistic in Cox proportional hazards survival models, with outcome `Y` and each row of `X` as covariate of interest, with possibly other covariates `Z.incl` from the matrix `Z`. | `boot` |
| `t.cor` | Pairwise correlation $t$-statistic for all variables in `X`. | `ic`$^\dagger$ |
| `z.cor` | Pairwise correlation Fisher's $z$-statistic for all variables in `X`. | `ic`$^\dagger$ |

Table 2: Available choices of test statistics implemented in the `MTP` and `EBMTP` functions in **multtest**. Supported corresponding test statistics null distributions are also given. For two-sample $t$-statistics and $F$-statistics, the class label is stored in the `Y` argument. The dagger symbol (†) denotes choices of test statistic (null distributions) which do not support weighted versions (i.e., `W` argument is ignored). Note that for `t.cor` and `z.cor`, the number of hypotheses can become large very fast.

| Slot | Value Description |
|---|---|
| `statistic` | Numeric vector of observed test statistics for each hypothesis. |
| `rawp` | Numeric vector of unadjusted, marginal $p$-values for each hypothesis. |
| `adjp` | Numeric vector of adjusted $p$-values for each hypothesis. |
| `reject` | Matrix of rejection indicators for each value of the nominal Type I error rate `alpha`. `TRUE` for a rejected null hypothesis. |
| `rawdist` | The numeric matrix for the estimated nonparametric bootstrap test statistics distribution. Returned only if `keep.rawdist=TRUE`. This slot must not be empty if one wishes to subsequently call `update` to update choice of bootstrap-based null distribution. |
| `nulldist` | The numeric matrix for the estimated test statistics null distribution, returned by default. Not available for the permutation distribution. If `rawdist` is empty, this slot must be returned if one wishes to subsequently call `update`. See software for details. |
| `nulldist.type` | Character value describing which choice of null distribution was used to generate the MTP results, i.e., the value of the `nulldist` argument in the function call. One of `‘boot.cs’`, `‘boot.ctr’`, `‘boot.qt’`, `‘ic’`, or `‘perm’`. |
| `marg.null` | If `nulldist=‘boot.qt’`, the character value of the marginal null distribution used by the MTP. Can be used to check default values or to ensure manual settings were correctly applied. See text for details. |
| `marg.par` | If `nulldist=‘boot.qt’`, a numeric matrix of parameters of the marginal null distribution(s) used by the MTP. Can be used to check default values or to ensure manual settings were correctly applied. See text for details. |

Table 3: Important common slots returned by objects of the classes *MTP* and *EBMTP*. Others slots are available, but they have been omitted in the context of the current discussion. See software for details.

| Argument | Description |
|---|---|
| `nulldist` | Character string indicating which method to use for estimating the joint test statistics null distribution. One of `‘boot.cs’`, `‘boot.ctr’`, `‘boot.qt’`, `‘perm’`, or `‘ic’`. |
| `csnull` | `DEPRECATED` |
| `marg.null` | If `nulldist=‘boot.qt’`, the marginal null distribution to use for quantile transformation. One of `‘normal’`, `‘t’`, `‘F’`, or `‘perm’`. Default is `NULL`, in which case the marginal null distribution is selected based on choice of test statistic. User can override defaults. |
| `marg.par` | If `nulldist=‘boot.qt’`, the parameters defining the marginal null distribution in `marg.null`. Default is `NULL`, in which case the values are selected based on choice of test statistic and/or other features (e.g., sample size, number of unique class labels, etc.). User can override defaults. |
| `ncp` | If `nulldist=‘boot.qt’`, a value for a possible noncentrality parameter to be used during marginal quantile transformation. Default is `NULL`. |
| `perm.mat` | If `nulldist=‘boot.qt’` and `marg.null=‘perm’`, a matrix of user-supplied test statistics from a distribution to be used during marginal quantile transformation. The statistics may be theoretically or empirically derived marginal permutation values. |
| `keep.margpar` | If `nulldist=‘boot.qt’`, a logical indicating whether the (internally created) matrix of marginal null distribution parameters should be returned. Default is `TRUE`. |
| `keep.nulldist` | Logical indicating whether to return the computed bootstrap or influence curve null distribution, by default `TRUE`. Not available for `nulldist=‘perm’`. |
| `keep.rawdist` | Replaces `csnull`. Logical indicating whether to return the computed raw, untransformed bootstrap distribution, by default `FALSE`. Not available for `nulldist=‘perm’` or `‘ic’`. |

Table 4: Amended or added arguments to the `MTP` and `EBMTP` function in **multtest** for controlling input and output of elements pertaining to the bootstrap-based null distributions in pkgmulttest. Note the matrix object returned in the slots `rawdist` and `nulldist` slots can become quite large.

| Test Statistic | Default Marginal Null Distribution |
|---|---|
| `t.onesamp` | $t$-distribution with $df = n - 1$ |
| `t.twosamp.equalvar` | $t$-distribution with $df = n - 2$ |
| `t.twosamp.unequalvar` | $\mathrm{N}(0, 1)$ |
| `t.pair` | $t$-distribution with $df = n - 1$, where $n$ is the number of unique samples, i.e., the number of observed differences between paired samples |
| `f` | $F$-distribution with $df_1 = k - 1$, $df_2 = n - k$, for $k$ groups |
| `f.twoway` | $F$-distribution with $df_1 = k - 1$, $df_2 = n - kl$, for $k$ groups and $l$ blocks |
| `lm.XvsZ` | $\mathrm{N}(0, 1)$ |
| `lm.YvsXZ` | $\mathrm{N}(0, 1)$ |
| `coxph.YvsXZ` | $\mathrm{N}(0, 1)$ |

Table 5: Default marginal null distributions for the quantile-transformed test statistics null distribution. User can override defaults by setting `marg.null` and `marg.par` manually.

| Argument | Description |
|---|---|
| `MVN.method` | Character string of either of `'mvrnorm'` (default) or `'Cholesky'` designating how correlated normal test statistics are to be generated. Selecting `'mvrnorm'` uses the function of the same name found in the `MASS` library [Venables and Ripley, 2002], whereas `'Cholesky'` relies on a Cholesky decomposition. |
| `penalty` | If `MVN.method='Cholesky'`, the value in `penalty` to be added to all diagonal elements of the estimated test statistics correlation matrix to ensure the matrix is positive definite. Guarantees internal calls to `chol` do not return an error. Default is `1e-6`. |
| `ic.quant.trans` | A logical indicating whether or not a marginal quantile tranformation using a $t$-distribution or user-supplied marginal distribution (stored in `perm.mat`) should be applied to the multivariate normal null distribution derived from the vector influence curve. Default is `'FALSE'`. Defaults for `marg.null` and `marg.par` exist, but values may also be specified by the user. See text and software for details. |

Table 6: Arguments to the `MTP` and `EBMTP` functions which control the implementation of the influence curve-based test statistics null distributions in the **multtest** software.

| Argument | Description |
|---|---|
| `typeone` | Character string indicating which Type I error rate to control, one of `'fwer'` (default), `'gfwer'`, `'tppfp'` or `'fdr'`. In particular, for control of gFWER, TPPFP or FDR, results are not obtained through augmentation as in `MTP`, but rather using guessed sets of true and false null hypotheses. |
| `method` | Character string indicating the EBMTP method. Currently only `'common.cutoff'`, which is most similar to `'ss.maxT'`. |
| `get.cr`, `get.cutoff`, `get.adjp` | `DEPRECATED` in `EBMTP`. Only adjusted $p$-values are available for EBMTPs. |
| `prior` | Character string indicating which choice of prior probability to use for estimating local $q$-values. Default is `'conservative'`, i.e., equal to one, meaning that all hypotheses are assumed to belong to the set of true null hypotheses. Relaxing the prior using `'ABH'` or `'EBLQV'`, for the adaptive Benjamini-Hochberg or empirical Bayes prior estimates, respectively, may result in more rejections, albeit at a cost of Type I error control under certain conditions. See text and references. |
| `bw, kernel` | Character string arguments to `density` indicating the smoothing bandwidth and kernel to be used during kernel density estimation. Defaults are `'nrd'` and `'gaussian'`, respectively. |
| `keep.falsepos`, `keep.truepos` | Logicals indicating whether to store the matrix of guessed false (true) positives obtained over all rounds of (re)sampling. |
| `keep.errormat` | A logical indicating whether to store the matrix of guessed Type I error rate values over all rounds of (re)sampling. In the case of FDR-control, for example, this matrix is `falsepos`/(`falsepos` + `truepos`). The row means of this matrix are used for ordering and assigning adjusted $p$-values to test statistics for each hypothesis. |
| `keep.Hsets` | A logical indicating whether to return the matrix of binary indicators partitioning the hypotheses into guessed sets of true and false null hypotheses obtained over all rounds of (re)sampling. |

Table 7: Arguments to the `EBMTP` function which distinguish it from the original `MTP` function in **multtest**.

| Slot | Value Description |
|---|---|
| `falsepos` | A matrix of the numbers of guessed false positives when using the values of the observed test statistics as cut-offs. Not returned unless `keep.falsepos=TRUE`. |
| `truepos` | A matrix of the numbers of guessed true positives when using the values of the observed test statistics as cut-offs. Not returned unless `keep.truepos=TRUE`. |
| `errormat` | The matrix obtained after applying the Type I error rate closure to the matrices in `falsepos`, and, if applicable, `truepos`. Not returned unless `keep.errormat=TRUE`. |
| `EB.h0M` | The sum of the local $q$-values obtained after density estimation. This number serves as an estimate of the proportion of true null hypotheses when `prior='EBLQV'`. Values close to one indicate situations in which Type I error control may not be guaranteed by the EBMTP. |
| `prior` | The numeric value of the prior probability $\pi_0$ used when evaluating the local $q$-value function of Equation (3). |
| `prior.type` | Character string returning the value of `prior` in the original call to `EBMTP`. One of `'conservative'`, `'ABH'`, or `'EBLQV'`. |
| `lqv` | A numeric vector of length equal to the number of hypotheses with elements containing the estimated local $q$-values used for generating guessed sets of true null hypotheses. |
| `Hsets` | A numeric matrix containing the Bernoulli realizations of the estimated local $q$-values stored in `lqv` which were used to partition the hypotheses into guessed sets of true and false null hypotheses at each round of (re)sampling. Not returned unless `keep.Hsets=TRUE`. |

Table 8: Additional slots returned by objects of class *EBMTP* relative to objects of class *MTP* in the package **multtest**.

| Procedure | Error Rate | Null Distribution | Rejections | Time |
|---|---|---|---|---|
| Bonferroni | FWER | N(0,1) | 23 | – |
| ss maxT | FWER | QT, N(0,1)* | 24 | 1h 12m 54s |
| ss maxT | FWER | NSS† | 14 | 7s |
| ss maxT | FWER | NS† | 9 | 6s |
| ss maxT | FWER | IC† | 31 | 4m 24s |
| EB, conservative | FWER | QT, N(0,1) | 31 | 1h 20m 18s |
| EB, conservative | FWER | QT, N(0,1)† | 30 | 7m 25s |
| EB, conservative | FWER | IC | 33 | 11m 42s |
| AMTP | gFWER, $k = 5$ | QT, N(0,1)† | 29 | 4s |
| EB, conservative | gFWER, $k = 5$ | QT, N(0,1)† | 74 | 7m 25s |
| AMTP | TPPFP, $q = 0.10$ | QT, N(0,1)† | 26 | 4s |
| EB, conservative | TPPFP, $q = 0.10$ | QT, N(0,1)† | 92 | 13m 52 |
| BH | FDR | N(0,1) | 130 | – |
| ABH | FDR | N(0,1) | 133 | – |
| TSBH, $\alpha = 0.05$ | FDR | N(0,1) | 133 | – |
| EB, conservative | FDR | QT, N(0,1)† | 148 | 13m 52s |
| EB, prior=ABH | FDR | QT, N(0,1)† | 159 | 13m 57s |
| EB, prior=EBLQV | FDR | QT, N(0,1)† | 161 | 14m 05s |

Table 9: Multiple testing results for tests of differential gene expression using the dataset of Chiaretti et al. [2004]. The number of rejections obtained by each MTP when controlling at nominal level $\alpha = 0.05$ are shown. Run times for joint MTPs are also given (applied to the filtered set of 2051 genes, using `B=5000` samples of null test statistics). Joint multiple testing results obtained using calls to `update` or `EBupdate` are denoted by the daggers symbol (†). The original null distribution used by the updated MTPs was the bootstrap-based null quantile-transformed test statistics joint null distribution with N(0,1) marginal distributions (row 2, denoted by *). EBMTP results were available after using the class conversion method `mtp2ebmtp`. Other choices of null distribution included NSS (null shifted and scaled, `'boot.cs'`), NS (null shifted, `'boot.ctr'`), and IC (influence curve, `'ic'`). In the case of EB FWER, the updated results are similar to those obtained from direct calls to `EBMTP`. For all choices of Type I error rate, EBMTPs rejected more hypotheses than their vanilla (marginal or joint) MTP counterparts.

## References

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57(1):289–300, 1995.

Y. Benjamini and Y. Hochberg. On the adaptive control of the false discovery rate in multiple testing with independent statistics. *Journal of Educational and Behavorial Statistics*, 25(1): 60–83, Spring 2000.

Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165–1188, 2001.

Y. Benjamini, A. M. Krieger, and D. Yekutieli. Adaptive linear step-up procedures that control the false discovery rate. *Biometrika*, 93(3):491–507, 2006.

C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. In *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, pages 3–62. 1936.

S. Chiaretti, X. Li, R. Gentleman, A. Vitale, M. Vignetti, F. Mandelli, J. Ritz, and R. Foa. Gene expression profiles of adult T-cell acute lymphoblastic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103(7):2771–2778, 2004. Data publicly available in the Bioconductor experimental data package **ALL**, `http://www.bioconductor.org`.

G. A. Churchill and R. W. Doerge. Empirical threshold values for quantitative trait mapping. *Genetics*, 138:963–971, 1994.

S. Dudoit and M. J. van der Laan. *Multiple Testing Procedures and Applications to Genomics.* Springer, 2008.

S. Dudoit, M. J. van der Laan, and K. S. Pollard. Multiple testing. Part I. Single-step procedures for control of general type I error rates. *Statistical Applications in Genetics and Molecular Biology*, 3(1), 2004. URL `http://www.bepress.com/sagmb/vol3/iss1/art13`.

S. Dudoit, H. N. Gilbert, and M. J. van der Laan. Resampling-based empirical bayes multiple testing procedures for controlling generalized tail probability and expected value error rates: Focus on the false discovery rate and simulation study. *Biometrical Journal*, 50(5):716–44, 2008. URL `http://www.stat.berkeley.edu/~houston/BJMCPSupp/BJMCPSupp.html`.

B. Efron, R. Tibshirani, J. D. Storey, and V. Tusher. Empirical Bayes analysis of a microarray experiment. *Journal of the American Staistical Association*, 96(456):1151–1160, 2001.

L. Gautier, L. Cope, B. M. Bolstad, and R. A. Irizarry. **affy**—analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004.

R. Gentleman and Biocore. ***geneplotter**: Graphics related functions for Bioconductor*, 2009. R package version 1.21.3.

R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. URL `http://genomebiology.com/2004/5/10/R80`.

H. N. Gilbert, M. J. van der Laan, and S. Dudoit. Joint multiple testing procedures for inferring lower-order conditional independence graphs with applications to biological networks. *In preparation*, 2009.

F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. URL `http://www.stat.uni-muenchen.de/~leisch/Sweave`. ISBN 3-7908-1517-9.

E. Neuwirth. ***RColorBrewer**: ColorBrewer palettes*, 2007. R package version 1.0-2.

K. S. Pollard and M. J. van der Laan. Choice of a null distribution in resampling-based multiple testing. *Journal of Statistical Planning and Inference*, 125:85–100, 2004.

K. S. Pollard, H. N. Gilbert, Y. Ge, S. Taylor, and S. Dudoit. ***multtest**: Resampling-Based Multiple Hypothesis Testing*, 2005. R package version 1.23.5.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

T. Schweder and E. Spjøtvoll. Plots of $p$-values to evaluate many tests simultaneously. *Biometrika*, 69:493–502, 1982.

J. D. Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society, Series B*, 64(3):479–498, 2002.

J. D. Storey and R. Tibshirani. Statistical significance for genomewide studies. *Proceedings of the National Academy of Science*, 100(16):9440–9445, 2003.

S. L. Taylor and K. S. Pollard. Hypothesis tests for point-mass mixture data with applications to 'omics data with many zero values. *Statistical Applications in Genetics and Molecular Biology*, 8(1), 2009. URL `http://www.bepress.com/sagmb/vol8/iss1/art8`.

S. L. Taylor, D. T. Lang, and K. S. Pollard. Improvements to the multiple testing package multtest. *R News*, 7(3):52–55, 2007.

R. Tibshirani, G. Chu, T. Hastie, and B. Narasimhan. ***samr**/SAM: Significance Analysis of Microarrays*, 2001. URL `http://www-stat.stanford.edu/~tibs/SAM`. R package version 1.26.

L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova. ***snow**: Simple Network of Workstations*, 2006. R package version 0.3-3.

V. Tusher, R. Tibshirani, and C. Chu. Significance analysis of microarrays applied to ionizing radiation response. *Proceedings of the National Academy of Sciences*, 98:5116–5121, 2001.

M. J. van der Laan and A. E. Hubbard. Quantile-function based null distribution in resampling based multiple testing. *Statistical Applications in Genetics and Molecular Biology*, 5(1), 2006. URL `http://www.bepress.com/sagmb/vol5/iss1/art14`.

M. J. van der Laan, S. Dudoit, and K. S. Pollard. Augmentation procedures for control of the generalized family-wise error rate and tail probabilities for the proportion of false positives. *Statistical Applications in Genetics and Molecular Biology*, 3(1), 2004a. URL `http://www.bepress.com/sagmb/vol5/iss1/art15`.

M. J. van der Laan, S. Dudoit, and K. S. Pollard. Multiple testing. Part II. Step-down procedures for control of the family-wise error rate. *Statistical Applications in Genetics and Molecular Biology*, 3(1), 2004b. URL `http://www.bepress.com/sagmb/vol5/iss1/art14`.

M. J. van der Laan, M. D. Birkner, and A. E. Hubbard. Empirical Bayes and resampling based multiple testing procedure controlling tail probability of the proportion of false positives. *Statistical Applications in Genetics and Molecular Biology*, 4(1), 2005. URL `http://www.bepress.com/sagmb/vol4/iss1/art29`.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S.* Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

A. von Heydebreck, W. Huber, and R. Gentleman. Differential expression with the bioconductor project. Technical Report 7, Bioconductor Project Work Papers, 2004. URL `http://www.bepress.com/biconductor/paper7`.

P. H. Westfall and S. S. Young. *Resampling-Based Multiple Testing: Examples and Methods for p-value Adjustment.* John Wiley and Sons, 1993.

D. Yekutieli and Y. Benjamini. Resampling-based false discovery rate controlling multiple test procedures for correlated test statistics. *Journal of Statistical Planning and Inference*, 82:171–196, 1999.