RSNPset: An Open R Package for Genome-Wide SNP Set Analysis on the Basis of Efficient Scores

November 19, 2015

Abstract

This package offers the capability to conduct genome-wide inference for case-control studies, including studies with censored phenotypes, using efficient score statistics. This document is intended to supplement the package and function documentation, to more thoroughly address finer statistical points encountered when applying this approach.

1 Introduction

Analysis of high-throughput genomic assay data on the basis of the score is asymptotically equivalent to Wald or likelihood ratio tests, offers higher computational efficiency, greater stability, and more readily lends itself to the use of resampling methods. Though the estimation of unknown nuisance parameters may induce variability on the score, the use of the efficient score accounts for this.

The RSNPset package provides a software implementation of efficient score statistics in genome-wide SNP set analysis of complex traits. This document provides an overview of the package, its methods for statistical inference, some example usage, and an explanation of the statistical assumptions of its arguments and options. By way of example, the SNP sets discussed here use genes as the loci of interest, i.e. they are sets of SNPs relevant to specific genes, but the approach is suitable for other genomic loci, including pathways and bands.

2 Score Calculations

We wish to test the association of a set of SNPs with a quantitative trait. We begin by establishing some notation. Let the number of patients be denoted by n (indexed by i), and the number of SNPs in a set be denoted m (indexed by j). The complete set of genotypes for all patients forms the matrix G, where the genotype for SNP j in patient i is denoted by G_{ij}. The vector of outcomes for all patients is denoted Y, where the outcome for patient i is denoted by Y_i. (Or, for right-censored endpoints, what is observed is (Y_i, Δ_i), where Y_i is the event time and $\Delta_i \in \{0, 1\}$ is the event indicator). The model may also include X, an $n \times p$ matrix of covariates, where $\vec{X}_i$ is the vector of covariate values for patient i.

The marginal null hypothesis for SNP j (that the variant is not associated with the outcome) is denoted by H_j. This hypothesis is tested using an efficient score statistic whose numerator is of the form $\sum_{i=1}^{n} U_{ij}$, where U_{ij} denotes the contribution of patient i to the efficient score [3] statistic. Table 1 shows the specific equations for U_{ij} for different types of outcomes.

Outcome	Score	U_{ij}
$Y_i \in \{0, 1\}$	Binomial	$(Y_i - \bar{Y})(G_{ij} - \bar{G}_j)$
$Y_i \in \mathbb{R}$	Gaussian	$(Y_i - \vec{X}_i\hat{\alpha})(G_{ij} - \vec{G}_j^T X(X^T X)^{-1}\vec{X}_i^T)$
$Y_i \in (0, \infty), \Delta_i \in \{0, 1\}$	Cox	$\Delta_i(G_{ij} - a_j(i)b(i)^{-1})$

Table 1: Summary of efficient score calculations. Here $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$ and $\bar{G}_j = \frac{1}{n}\sum_{i=1}^{n} G_{ij}$. For the Gaussian case, $\vec{G}_j$ is the vector of genotypes for SNP j for all patients, and $\hat{\alpha}$ is the MLE estimator for the coefficients of the covariates, estimated from a model without the genotypic component. Note that in the absence of covariates, the design matrix can be considered a vector of 1's and the Gaussian equation reduces to the binomial case. For the Cox score, $a_j(i) = \sum_{l=1}^{n} \mathbb{I}[Y_l \geq Y_i]G_{lj}$ and $b(i) = \sum_{l=1}^{n} \mathbb{I}[Y_l \geq Y_i]$.

Each of the K SNP sets, J_k, are composed of m_k SNPs, and the null hypothesis for each SNP set is denoted by $\mathbb{H}_k = \cap_j H_j$ where $j \in J_k$. To derive the efficient score for the SNP set, we construct $\boldsymbol{U}_k$, the $n \times m_k$ matrix whose (i, j) element is U_{ij}. Then $\boldsymbol{U}_{\bullet k} = (\sum_{i=1}^{n} U_{i1}, \ldots, \sum_{i=1}^{n} U_{im_k})^T$ is the corresponding vector of the m_k marginal score statistics, and $\boldsymbol{\Sigma}_k = \boldsymbol{U}_k^T\boldsymbol{U}_k$ is the corresponding covariance matrix. The efficient score statistic for the SNP set hypothesis $\mathbb{H}_k$ then is $W_k = \boldsymbol{U}_{\bullet k}^T\boldsymbol{\Sigma}_k^{+}\boldsymbol{U}_{\bullet k}$, where $\boldsymbol{\Sigma}_k^{+}$ is the Penrose-Moore inverse of $\boldsymbol{\Sigma}_k$.

Under suitable regularity conditions [1], W_k converges in distribution to a chi-squared distribution with degrees of freedom $\nu = R_k \in \{1, \ldots, m_k\}$ and centrality parameter δ, denoted by $\chi^2[\nu, \delta]$. As R_k is an unknown parameter, under $\mathbb{H}_k$ we approximate the null distribution of W_k as $\chi^2[r_k, 0]$, where r_k is the rank of $\boldsymbol{\Sigma}_k$. (Note that the maximum possible rank of $\boldsymbol{\Sigma}_k$ for any SNP set is $n - 1$). Thus we are able to compute the efficient score and asymptotic p-value for each of the K SNP sets.

The package includes options for two different resampling methods. The first is Monte Carlo resampling [2]. We obtain a replicate of $\{(W_1, r_1), \ldots, (W_K, r_K)\}$, call it $\{(W_{b,1}, r_{b,1}), \ldots, (W_{b,K}, r_{b,K})\}$, by replacing $\boldsymbol{U}_{\bullet k} = U_k^T\vec{1}_n$ above with $\boldsymbol{U}_{\bullet k} = U_k^T\vec{Z}_n$, where $\vec{Z}_n$ denotes a vector of length n of standard normal random values. Alternatively, we can generate $\{(W_{b,1}, r_{b,1}), \ldots, (W_{b,K}, r_{b,K})\}$ by permuting the outcome vector, $(Y_1, \ldots, Y_n)$, while holding the genotype matrix for each SNP set unchanged. In the case of right-censored endpoints, (Y_i, δ_i) pairs are permuted intact. Note that the permutation method is not appropriate when the model includes covariates. These resampling replicates are useful in assessing the significance of the statistical results.

2.1 Notes on `rspset()`

Users should be aware that this function does not run checks to confirm that the elements of a requested SNP set are present in the matrix `G` before executing. No results are returned for SNP sets for which no SNPs are present or which include **any** SNPs with **missing** data values. If a SNP set contains column names that are not present in the argument `G`, the function executes without objection and returns a test statistic based on the subset of columns that *are* present. For this reason, the `summary.RSNPset()` function provides (in addition to other execution information) counts of the number of SNP sets dropped from the analysis and the number of SNP sets defined to include SNPs that are not present in the data, if any.

Unlike the Gaussian and bionomial score statistics, the Cox score statistic (`method = "cox"`) is not a sum of independent patient level scores, so some level of pruning of SNPs is recommended. The permutation resampling facilities of the package can be utilized to assess the performance of the asymptotic inference. The development of robust methods for calculating the Cox score statistics and approximating the covariance matrix is under way.

2.2 Options for `rspset()`

The following options imply specific assumptions about the model or data. It is important to confirm that these assumptions are valid when executing the analysis.

`X` - Permutation resampling is inappropriate for models including covariates (`r.method` must be set as `"monte carlo"`). Currently, the inclusion of covariates is implemented only for the Gaussian model.

`v.permute` - When using `r.method = "permutation"`, setting this value to `FALSE` saves processing time by not re-calculating the variance of the permutation replicates, under the assumption that $\mathbf{\Sigma}_k$ for SNP set J_k is the same across permutations.

`ret.rank` - When using `r.method = "permutation"`, setting this option to `FALSE` reduces the size of the returned object by not returning ranks for the permutation replicates, under the assumption that the rank of $\mathbf{\Sigma}_k$ for SNP set J_k is invariant under permutation. Ranks are returned only for the observed data.

`pinv.check` - The calculation of the efficient score relies on the Penrose-Moore inverse of the variance matrices. Setting this argument to `TRUE` generates a list of `B+1` data frames, each with one row per SNP set and five columns of diagnostic measures of the accuracy of this inverse. Departure of these values from zero indicates poor performance of the Penrose-Moore inverse. These diagnostics are returned as an attribute that can be accessed via the `summary()` function.

3 Hypothesis Testing

At minimum, the `rspset.pvalue()` function returns asymptotic p-values and false discovery rate (FDR) adjusted q-values for each SNP set. Setting the `rspset()` argument $\texttt{B} > 0$ allows `rspset.pvalue()` to return resampling p-values (and FDR adjusted resampling q-values) as well. By default, the `rspset.pvalue()` function uses the `qvalue` package to compute the q-values, though the `qfun` argument can be used to provide a custom definition.

3.1 Notes on `rspset.pvalue()`

Several considerations need to be made in testing the association of a SNP set with the outcome. Firstly, meaningful unadjusted resampling p-values require $\texttt{B} > K/\alpha$ (on the order of 10^7 replicates for a genome-wide study) in order to account for false positives due to multiple testing. Note that in the event that none of the resampling replicates generates a more extreme result than the observed value, a resampling p-value of $1/\texttt{B}$ is returned (instead of 0).

Second, by default `rspset()` and `rspset.pvalue()` operate in accordance with the assumption that each $\mathbf{\Sigma}_k$ is invariant under resampling, in which case the degrees of freedom of the chi-squared distribution for the statistics, W_k, are the same across replications. This is always true in the case of Monte Carlo resampling, but may not hold for permutation resampling. If the assumption is valid, then the observed and replication statistics for a SNP set are directly comparable. However, if the ranks of the $\mathbf{\Sigma}_k$ differ across permutations, then the chi-squared distributions differ in degrees of freedom, so the raw statistics are not comparable. In this latter case, the arguments of `rspset()` should be set to return the ranks of the permutation replicates (`ret.rank = TRUE`), and `rspset.pvalue()` should be run with the `pval.transform` argument set to `TRUE`. When `pval.transform = TRUE`, instead of comparing the observed and resampling statistics, the resampling p-values are determined by comparing observed asymptotic p-values to the asymptotic p-values of the resampling replicates.

3.2 Options for `rspset.pvalue()`

`pval.transform` - As mentioned above, when this option is set to `TRUE`, the function uses the ranks of the resampling replicates to get resampling replicate p-values (which are identically distributed, and thus comparable) to determine the empirical resampling p-values of the statistics.

`qfun` - The `qvalue()` function, which is used internally in this function, may fail when used on a small number of replications or SNP sets. This argument is used to define a new q-value function, or to assign arguments for the `qvalue()` function. For example, `qfun = function(x){qvalue(x, robust = TRUE)$qvalue}` can be used to avoid generating errors if the number of SNP sets is small.

4 An Example Analysis

In practice, analyses might include hundreds of patients and thousands of SNP sets, spanning tens of thousands of SNPs. For the purposes of this demonstration, we simulate a more manageable example. Users should also note that an important precursor to genome-wide analyses is quality control of the genotypic data. As our data is simulated and complete, we can omit this step and proceed with our analysis.

4.1 Simulating the Data

First, we generate a cohort of `n` patients and their outcome data, i.e., the traits we wish to analyze: case-control status, LDL level, and survival time/survival status.

```
set.seed(123)

n <- 100
status <- rbinom(n, 1, 0.5)
table(status)

## status
##  0  1
## 53 47

LDL <- rnorm(n, mean=115, sd=35)
quantile(LDL)

##        0%       25%       50%       75%      100%
##  34.17909  91.86624 113.23968 133.43291 191.55665

time <- rexp(n, 1/10)
event <- rbinom(n, 1, 0.9)
quantile(time)

##         0%        25%        50%        75%       100%
##  0.2417337  3.6029326  7.6329649 15.4336484 43.6591094

table(event)

## event
##  0  1
## 10 90
```

We also simulate some covariates to include in the analysis of our continuous phenotype. Note that the covariates must be numeric.

```
X <- matrix(c(rnorm(n), rbinom(n, 4, .25)), nrow=n)
summary(X[,1])


##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.466000 -0.716800  0.027770 -0.002918  0.716800  2.571000


table(X[,2])


##
##  0  1  2  3
## 36 44 16  4
```

Next we simulate the genomic data. Here we are using allele counts, but the methodology is also applicable to expression levels. For each SNP, we let the probability of having a mutant allele be a random value selected from a uniform distribution across the interval $(0.1, 0.9)$.

```
m <- 500

G <- matrix(as.double(rbinom(n*m, 2, runif(n*m,.1,.9))), n, m)
dim(G)


## [1] 100 500
```

We must label all of the SNPs so that they can be referenced by the SNP sets. The rows correspond to the genotypes of each patient, and the columns are the allele counts for each SNP.

```
rsIDs <- paste0("rs100",1:m)
colnames(G) <- rsIDs

G[1:5,1:5]


##      rs1001 rs1002 rs1003 rs1004 rs1005
## [1,]      2      0      0      1      2
## [2,]      1      2      0      2      2
## [3,]      2      0      2      0      1
## [4,]      2      2      2      1      0
## [5,]      0      1      2      2      2
```

Next we make the SNP sets. For our example we compose $K = 10$ sets of between three and fifty SNPs at random, but we imagine them to represent groups of SNPs related to specific genes (or perhaps some other genomic loci, such as bands or pathways).

```
K <- 10
genes <- paste0("XYZ",1:K)
geneSets <- lapply(sample(3:50, size=K, replace=TRUE), sample, x=rsIDs)
names(geneSets) <- genes

unlist(lapply(geneSets, length))


##  XYZ1  XYZ2  XYZ3  XYZ4  XYZ5  XYZ6  XYZ7  XYZ8  XYZ9 XYZ10
##    37    49    34    36    43    46    16    34    47    26
```

For a systematic approach to generating gene-based SNP sets from real genomic data, see the `snplist` [4] package.

4.2 Conducting the Analysis

After installing its dependent packages, we load `RSNPset`. The fact that `RSNPset` utilizes the `doRNG` package gives us the ability to set a seed so that our analyses are reproducible.

```
library(RSNPset)
set.seed(456)
```

We look first at our binary phenotype, case-control status. Again, a dramatically greater number of replications is generally required in order to attain meaningful results. We use just a few here for the purposes of demonstration.

```
ccres <- rsnpset(Y=status, G=G, snp.sets=geneSets, score="binomial",
                 B=10, r.method="permutation", ret.rank=TRUE, v.permute=TRUE)


## Loading required package:  foreach
## Loading required package:  rngtools
## Loading required package:  pkgmaker
## Loading required package:  registry
##
## Attaching package:  'pkgmaker'
##
## The following object is masked from 'package:base':
##
##    isNamespaceLoaded
```

The resulting object, `ccres`, includes a list of `B+1` data frames. Each data frame has one row per (non-empty) SNP set and a column, `W`, containing the test statistics. The first data frame contains the statistics for the observed data, while the remainder correspond to the permutation results. Since we set `ret.rank = TRUE`, each data frame also contains a column with the rank of the covariance matrix, $\boldsymbol{\Sigma}_k$, i.e. the degrees of freedom for our chi-squared test. The first data frame also contains a column, `m`, denoting the number of SNPs in each SNP set.

```
ccres[["Observed"]]
```

```
##              W rank  m
## XYZ1  26.60494   37 37
## XYZ2  46.48412   49 49
## XYZ3  44.80248   34 34
## XYZ4  29.61905   36 36
## XYZ5  42.96863   43 43
## XYZ6  39.06485   46 46
## XYZ7  14.71040   16 16
## XYZ8  43.52848   34 34
## XYZ9  52.97704   47 47
## XYZ10 35.80744   26 26
```

The `summary()` function displays the execution parameters for the returned object.

```
summary(ccres)
```

```
## - Efficient score statistics based on 100 samples.
## - SNP sets range in size from 16 to 49.
## - 0 SNP sets were not included in the analysis
## - 0 SNP sets contained SNPs that were not included in the analysis.
## - 10 permutation replicates were computed.
## - ret.rank = TRUE : The ranks of the permutation variance matrices were returned.
## - v.permute = TRUE : Variance was recomputed for each permutation replicate.
## [1] NA
```

We use `rspset.pvalue()` to check for evidence for rejecting our null hypotheses. Since we have the ranks for the permutation replicates, we set `pval.transform = TRUE` to ensure the results across permutations are comparable.

```
rsnpset.pvalue(ccres, pval.transform=TRUE)
```

```
##              W rank  m          p  pB  PB          Q         QB
## XYZ1  26.60494   37 37 0.89709246 1.0 1.0 0.2343892 1.0000000
## XYZ2  46.48412   49 49 0.57570872 0.8 1.0 0.2148845 1.0000000
## XYZ3  44.80248   34 34 0.10179386 0.1 0.3 0.1104441 0.3333333
## XYZ4  29.61905   36 36 0.76478844 0.9 1.0 0.2220236 1.0000000
## XYZ5  42.96863   43 43 0.47265895 0.4 1.0 0.2148845 0.6666667
## XYZ6  39.06485   46 46 0.75567628 0.7 1.0 0.2220236 1.0000000
## XYZ7  14.71040   16 16 0.54594305 0.3 1.0 0.2148845 0.6000000
## XYZ8  43.52848   34 34 0.12681291 0.1 0.6 0.1104441 0.3333333
## XYZ9  52.97704   47 47 0.25454147 0.3 0.9 0.1662643 0.6000000
## XYZ10 35.80744   26 26 0.09526779 0.1 0.3 0.1104441 0.3333333
```

As explained above, setting `pval.transform = TRUE` means the function compares asymptotic p-values across permutations, instead of raw statistics, in computing the resampling p-values (`pB`). Having computed these p-values, the function also returns family-wise error adjusted p-values (`PB`), in addition to the asymptotic (`p`) and FDR adjusted p-values (`Q` and `QB`). As expected, since the data are simulated, we find no significant association between any SNP sets and the outcome.

We move on to the continuous phenotype, LDL level. Here we revert to the default values for the arguments `ret.rank` and `v.permute`. In practice, when used on a much larger set of data, these changes offer the potential for a significant reduction in both the processing time and the size of the returned object. The default `r.method`, Monte Carlo resampling, allows us to include covariates in our model.

```
ldlres <- rsnpset(Y=LDL, G=G, X=X, snp.sets=geneSets, score="gaussian", B=10)
```

Since the default is `ret.rank = FALSE`, only the first data frame contains the column with the ranks of the covariance matrices.

```
ldlres[["Observed"]]

##              W rank  m
## XYZ1  44.65202   37 37
## XYZ2  42.53299   49 49
## XYZ3  38.49372   34 34
## XYZ4  41.23972   36 36
## XYZ5  39.52648   43 43
## XYZ6  43.72532   46 46
## XYZ7  15.73798   16 16
## XYZ8  44.80231   34 34
## XYZ9  46.37713   47 47
## XYZ10 19.10642   26 26
```

```
ldlres[["Replication.1"]]
```

```
##               W
## XYZ1  34.65214
## XYZ2  50.05806
## XYZ3  37.00862
## XYZ4  29.37871
## XYZ5  37.13304
## XYZ6  39.64480
## XYZ7  14.57184
## XYZ8  29.67163
## XYZ9  48.72751
## XYZ10 21.58943
```

Here the statistics of the Monte Carlo replicates have the same degrees of freedom as the observed statistics.

```
rsnpset.pvalue(ldlres)
```

```
##               W rank  m         p  pB           Q        QB
## XYZ1  44.65202   37 37 0.1811392 0.2 0.008360634 0.4000000
## XYZ2  42.53299   49 49 0.7311408 0.7 0.009939210 0.7777778
## XYZ3  38.49372   34 34 0.2733417 0.2 0.008360634 0.4000000
## XYZ4  41.23972   36 36 0.2521873 0.1 0.008360634 0.4000000
## XYZ5  39.52648   43 43 0.6227411 0.5 0.009523813 0.6250000
## XYZ6  43.72532   46 46 0.5680106 0.2 0.009523813 0.4000000
## XYZ7  15.73798   16 16 0.4713929 0.4 0.009523813 0.5714286
## XYZ8  44.80231   34 34 0.1017968 0.1 0.008360634 0.4000000
## XYZ9  46.37713   47 47 0.4982542 0.4 0.009523813 0.5714286
## XYZ10 19.10642   26 26 0.8319055 1.0 0.010178114 1.0000000
```

Again, these resampling p-values (`pBk`) are computed using identical covariance matrices, so the observed and replication statistics are directly comparable.

Lastly, we look at our right-censored (time to event) phenotype.

```
tteres <- rsnpset(Y=time, delta=event, G=G, snp.sets=geneSets, score="cox",
                  B=10, r.method="permutation", pinv.check=TRUE)
```

By setting `pinv.check = TRUE`, the returned object now includes an attribute giving five diagnostic measures of the Penrose-Moore Inverse computed for each SNP set in all of the permutation replicates and the observed data. These are accessed via the `summary()` function. The list of (`B`+1) $K \times 5$ data frames can be captured for examination.

```
pinv.diag <- summary(tteres)

## - Efficient score statistics based on 100 samples.
## - SNP sets range in size from 16 to 49.
## - 0 SNP sets were not included in the analysis
## - 0 SNP sets contained SNPs that were not included in the analysis.
## - 10 permutation replicates were computed.
## - ret.rank = FALSE : The ranks of the permutation variance matrices were not returned
## - v.permute = FALSE : Variance was not recomputed for each permutation replicate.
## - pinv.tol = 7.8e-08

pinv.diag[["Observed"]]

##              d0           d1           d2           d3           d4
## 1  1.728750e-11 1.720935e-11 6.973588e-15 3.312108e-14 3.307164e-14
## 2  1.388401e-11 1.377742e-11 7.928554e-15 1.515259e-13 1.514843e-13
## 3  5.778933e-12 5.789813e-12 3.705153e-15 1.921553e-14 1.908803e-14
## 4  3.261391e-12 3.396394e-12 3.955170e-16 1.608258e-15 1.562986e-15
## 5  3.264500e-12 3.228084e-12 3.249137e-15 2.478226e-14 2.515869e-14
## 6  2.529532e-12 2.499334e-12 1.025569e-14 5.811324e-15 5.714179e-15
## 7  2.139267e-11 2.140865e-11 9.054389e-15 4.756091e-14 4.748979e-14
## 8  1.303180e-12 1.296352e-12 1.142532e-15 5.329764e-14 5.325644e-14
## 9  9.320433e-12 9.332757e-12 9.900067e-15 3.131870e-14 3.123890e-14
## 10 3.964828e-12 4.135359e-12 2.529227e-15 6.425199e-15 6.431440e-15

unlist(lapply(pinv.diag, max))

##       Observed  Replication.1  Replication.2  Replication.3
##   2.140865e-11   2.977174e-11   2.311396e-11   3.468870e-11
##  Replication.4  Replication.5  Replication.6  Replication.7
##   2.162004e-11   2.454748e-11   3.292611e-11   2.973621e-11
##  Replication.8  Replication.9 Replication.10
##   2.987122e-11   2.534506e-11   2.363976e-11
```

The diagnostic measures are all close to zero, indicating the Penrose-Moore Inverses are performing well, so we proceed to examining the p-values.

```
ttepvals <- rsnpset.pvalue(tteres)
ttepvals

##              W rank  m         p  pB         Q QB
## XYZ1  25.68844   37 37 0.9192003 0.9 0.7648177  1
## XYZ2  46.31991   49 49 0.5824286 0.7 0.7648177  1
## XYZ3  34.07853   34 34 0.4639620 0.3 0.7648177  1
## XYZ4  32.97182   36 36 0.6133919 0.8 0.7648177  1
## XYZ5  36.39498   43 43 0.7515701 0.8 0.7648177  1
## XYZ6  45.11153   46 46 0.5093921 0.8 0.7648177  1
## XYZ7  18.25057   16 16 0.3094295 0.3 0.7648177  1
## XYZ8  24.65034   34 34 0.8800991 0.9 0.7648177  1
## XYZ9  38.26542   47 47 0.8142634 1.0 0.7648177  1
## XYZ10 29.93221   26 26 0.2704289 0.4 0.7648177  1
```

A `summary()` method is also available for the results of `rspset.pvalue()`, by default returning the ten SNP sets with the smallest asymptotic p-values. The `verbose = TRUE` option gives additional information about the calculations.

```
summary(ttepvals, verbose=TRUE)
```

```
##
## - Resampling p-values (pB) come from comparison of
## test statistics across 10 replications.
## - Q-values based on 10 SNP sets.
##              W rank  m         p  pB         Q QB
## XYZ10 29.93221   26 26 0.2704289 0.4 0.7648177  1
## XYZ7  18.25057   16 16 0.3094295 0.3 0.7648177  1
## XYZ3  34.07853   34 34 0.4639620 0.3 0.7648177  1
## XYZ6  45.11153   46 46 0.5093921 0.8 0.7648177  1
## XYZ2  46.31991   49 49 0.5824286 0.7 0.7648177  1
## XYZ4  32.97182   36 36 0.6133919 0.8 0.7648177  1
## XYZ5  36.39498   43 43 0.7515701 0.8 0.7648177  1
## XYZ9  38.26542   47 47 0.8142634 1.0 0.7648177  1
## XYZ8  24.65034   34 34 0.8800991 0.9 0.7648177  1
## XYZ1  25.68844   37 37 0.9192003 0.9 0.7648177  1
```

As a typical GWAS study may span thousands of SNPs and SNP sets, `summary()` allows for the succinct listing of p-values for the most significant results. The returned data frame can be saved for future reference or reporting.

```
ttesum <- summary(ttepvals, sort="pB", nrows=5, dropcols=c("m","Q","QB"))
ttesum
```

```
##              W rank         p  pB
## XYZ3  34.07853   34 0.4639620 0.3
## XYZ7  18.25057   16 0.3094295 0.3
## XYZ10 29.93221   26 0.2704289 0.4
## XYZ2  46.31991   49 0.5824286 0.7
## XYZ4  32.97182   36 0.6133919 0.8
```

References

[1] Andrews D.K.W. Asymptotic Results for Generalized Wald Tests. *Econometric Theory*, 3:348-358, 1987.

[2] Lin D. An Efficient Monte Carlo Approach to Assessing Statistical Significance in Genomic Studies. *Bioinformatics*, 21(6):781-787, 2005.

[3] Tsiatis A.A. *Semiparametric Theory and Missing Data*. Springer Science+Business Media, LLC, New York, NY, 2006.

[4] `snplist` is available under the GNU General Public License from the Comprehensive R Archive Network (CRAN) web site.