Implementing a Class of Permutation Tests: The coin Package

Torsten Hothorn
Ludwig-Maximilians-Universität München

Kurt Hornik
Wirtschaftsuniversität Wien

Mark A. van de Wiel
Vrije Universiteit Amsterdam

Achim Zeileis
Wirtschaftsuniversität Wien

Abstract

This description of the R package **coin** is a (slightly) modified version of Hothorn, Hornik, van de Wiel, and Zeileis (2008a) published in the *Journal of Statistical Software*.

The R package **coin** implements a unified approach to permutation tests providing a huge class of independence tests for nominal, ordered, numeric, and censored data as well as multivariate data at mixed scales. Based on a rich and flexible conceptual framework that embeds different permutation test procedures into a common theory, a computational framework is established in **coin** that likewise embeds the corresponding R functionality in a common S4 class structure with associated generic functions. As a consequence, the computational tools in **coin** inherit the flexibility of the underlying theory and conditional inference functions for important special cases can be set up easily. Conditional versions of classical tests—such as tests for location and scale problems in two or more samples, independence in two- or three-way contingency tables, or association problems for censored, ordered categorical or multivariate data—can easily be implemented as special cases using this computational toolbox by choosing appropriate transformations of the observations. The paper gives a detailed exposition of both the internal structure of the package and the provided user interfaces along with examples on how to extend the implemented functionality.

Keywords: conditional inference, exact distribution, conditional Monte Carlo, categorical data analysis, R.

1. Introduction

Conditioning on all admissible permutations of the data for testing independence hypotheses is a very old, yet very powerful and popular, idea (Fisher 1935; Ernst 2004). Conditional inference procedures, or simply *permutation* or *re-randomization* tests, are implemented in many different statistical computing environments. These implementations, for example `wilcox.test()` for the Wilcoxon-Mann-Whitney test or `mantelhaen.test()` for the Cochran-Mantel-Haenszel χ^2 test in R (R Development Core Team 2008) or the tools implemented in **StatXact** (Cytel Inc. 2003), **LogXact** (Cytel Inc. 2006), or Stata (StataCorp. 2003)—see Oster (2002, 2003) for an overview—all follow the classical classification scheme of inference procedures and offer procedures for location problems, scale problems, correla-

group	time											
control	300	300	300	300	300	300	300	300	300	300	300	300
treatment	18	22	75	163	271	300	300	300	300	300	300	300

Table 1: The `rotarod` data: length of `time` on rotating cylinder by `group`.

tion, or nominal and ordered categorical data. Thus, each test procedure is implemented separately, maybe with the exception of conditional versions of linear rank statistics (Hájek, Šidák, and Sen 1999) in `NPAR1WAY` as available in SAS (SAS Institute Inc. 2003).

Theoretical insights by Strasser and Weber (1999) open up the way to a unified treatment of a huge class of permutation tests. The **coin** package for conditional inference is the computational counterpart to this theoretical framework, implemented in the R system for statistical computing (R Development Core Team 2008). Hothorn, Hornik, van de Wiel, and Zeileis (2006) introduce the package and illustrate the transition from theory to practice. Here, we focus on the design principles upon which the **coin** implementation is based as well as on the more technical issues that need to be addressed in the implementation of such conceptual tools. Within package **coin**, formal S4 classes describe the data model and the conditional test procedures, consisting of multivariate linear statistics, univariate test statistics and a reference distribution. Thus, one can work with objects representing the theoretical entities nearly in the same way as one would work with mathematical symbols. Generic functions for obtaining statistics, conditional expectation and covariance matrices as well as p value, distribution, density and quantile functions for the reference distribution help to extract information from these objects. The infrastructure is conveniently interfaced in the function `independence_test()`, thus providing the computational counterpart of the theoretical framework of Strasser and Weber (1999).

Here, we start out with an illustrative application of `independence_test()` to a small 2-sample problem (see Table 1) and then continue to introduce the underlying computational building blocks using the same data set. The data was used previously by Bergmann, Ludbrook, and Spooren (2000) as a challenging example in a comparison of test statistics and p values of the Wilcoxon-Mann-Whitney rank sum test as reported by eleven statistical packages. More specifically, $n = 24$ rats received a fixed oral dose of a centrally acting muscle relaxant as active treatment or a saline solvent as control. The animals were placed on a rotating cylinder and the length of time each rat remained on the cylinder was measured, up to a maximum of 300 seconds. The rats were randomly assigned to the control and treatment groups, thus a re-randomization test as implemented in `independence_test()` is the appropriate way to investigate if the response is independent of the group assignment. The data are particularly challenging because of the many ties in the (censored) response (19 observations take the maximal value 300) and the quasi-complete separation (smaller values of time are only observed in the treatment group). Conceptually, this makes computation of the two-sided exact p value for any 2-sample contrast very simple: $p = 2 \cdot \binom{19}{12} / \binom{24}{12} = 0.03727$. However, in software, this often makes computations of the exact p value more difficult because several simple algorithms fail.

Utilizing **coin**, the hypothesis of independence of length of time and group assignment can be specified by a formula which, together with a data frame `rotarod`, serve as arguments to `independence_test`:

```
R> library("coin")
R> data("rotarod", package = "coin")
R> independence_test(time ~ group, data = rotarod,
+    ytrafo = rank, distribution = exact())

        Exact General Independence Test

data:  time by group (control, treatment)
Z = 2.4389, p-value = 0.03727
alternative hypothesis: two.sided
```

Here, the conditional Wilcoxon-Mann-Whitney test was performed via a rank transformation of the response, employing the exact distribution for obtaining the p value (yielding the correct result outlined above). Users of R can easily interpret this output since it is represented in the same format as classical tests in the basic **stats** package. Based on the p value derived from the exact conditional distribution of the test statistic Z, the independence of group assignment and time on the cylinder can be rejected.

Although the above piece of code looks embarrassingly simple, the underlying computations are much more sophisticated than visible at first sight: The data are pre-processed along with their transformations, deviations from independence are captured by a (possibly multivariate) linear statistic, standardized by conditional expectation and variance, and aggregated to a final test statistic. Subsequently, the requested reference distribution is computed, from which a p value is derived, and everything is wrapped into an object that can be conveniently printed or queried for more detailed information. After briefly reviewing the underlying theory from Strasser and Weber (1999) in Section 2, we introduce formal S4 classes and methods capturing all the outlined steps in Section 3. Section 4 provides further information about high-level user interfaces and extensibility, and Section 5 further illustrates how to employ the software in practice using a categorical data example. Section 6 concludes the paper with a short discussion, some more details about the underlying theory can be found in an appendix.

2. Permutation tests in a nutshell

In the following we give a brief overview of the general theory for permutation tests as developed by Strasser and Weber (1999) and implemented by Hothorn *et al.* (2006).

The task is to test the independence of two variables $\mathbf{Y}$ and $\mathbf{X}$ from sample spaces $\mathcal{Y}$ and $\mathcal{X}$ which may be measured at arbitrary scales and may be multivariate as well. In addition, $b \in \{1, \ldots, k\}$, a factor measured at k levels, indicates a certain block structure of the observations: for example study centers in a multi-center randomized clinical trial where only a re-randomization of observations within blocks is admissible. We are interested in testing the null hypothesis

$$H_0 : D(\mathbf{Y}|\mathbf{X}, b) = D(\mathbf{Y}|b)$$

of conditional independence of $\mathbf{Y}$ and $\mathbf{X}$ within blocks b against arbitrary alternatives, for example shift or scale alternatives, linear trends, association in contingency tables etc. Strasser and Weber (1999) suggest deriving scalar test statistics for testing H_0 from multivariate linear

statistics of the form

$$\mathbf{T} = \sum_{j=1}^{k} \mathbf{T}_j \in \mathbb{R}^{pq} \tag{1}$$

where the linear statistic for each block is given by

$$\mathbf{T}_j = \text{vec}\left(\sum_{i=1}^{n} I(b_i = j) w_i g(\mathbf{X}_i) h(\mathbf{Y}_i)^\top\right) \in \mathbb{R}^{pq}.$$

The function $I(\cdot)$ is the indicator function and vec denotes the vec operator (which stacks the columns of a matrix). Here, $g : \mathcal{X} \to \mathbb{R}^{p \times 1}$ is a transformation of the $\mathbf{X}$ measurements and $h : \mathcal{Y} \to \mathbb{R}^{q \times 1}$ is a transformation of the $\mathbf{Y}$ values. The function $h(\mathbf{Y}_i) = h(\mathbf{Y}_i, (\mathbf{Y}_1, \ldots, \mathbf{Y}_n))$ is also called *influence function* and may depend on the full vector of responses $(\mathbf{Y}_1, \ldots, \mathbf{Y}_n)$, however only in a permutation symmetric way, i.e., the value of the function must not depend on the order in which $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$ appear. The case weights w_i are assumed to be integer-valued, indicating that w_i observations with realizations $\mathbf{Y}_i$, $\mathbf{X}_i$ and b_i are available, with default $w_i \equiv 1$.

The distribution of $\mathbf{T}$ depends on the joint distribution of $\mathbf{Y}$ and $\mathbf{X}$, which is unknown under almost all practical circumstances. At least under the null hypothesis one can dispose of this dependency by fixing $\mathbf{X}_1, \ldots, \mathbf{X}_n$ and conditioning on all possible permutations of the responses $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$ within block $j, j = 1, \ldots, k$. The conditional expectation $\mu \in \mathbb{R}^{pq}$ and covariance $\Sigma \in \mathbb{R}^{pq \times pq}$ of $\mathbf{T}$ under H_0 given all permutations $\sigma \in S$ of the responses are derived by Strasser and Weber (1999) and are given in Appendix A. Having the conditional expectation and covariance at hand we are able to standardize an observed linear statistic $\mathbf{t} \in \mathbb{R}^{pq}$ (of the form given in Equation 1) and aggregate it to some univariate test statistic $c = c(\mathbf{t}, \mu, \Sigma)$. Various choices for $c(\mathbf{t}, \mu, \Sigma)$ are conceivable, e.g., a quadratic form or a maximum type statistic (see Section 3). In the latter case, a natural first step is to standardize each of the pq statistics in $\mathbf{t}$ by its expectation and standard deviation:

$$\mathbf{z} = \text{diag}(\Sigma)^{-1/2}(\mathbf{t} - \mu). \tag{2}$$

In the following, we describe a class structure for representing these theoretical objects along with possible choices of test statistics c and computations or approximations of the associated reference distributions.

3. A class structure for permutation tests

In this section, the theory of permutation tests, as briefly outlined in the previous section, is captured in a set of S4 classes and methods: In Section 3.1, we suggest objects representing the data and the independence problem, for which a test statistic is computed subsequently. Objects for the associated reference distribution are constructed in Section 3.2. Finally, in Section 3.3, everything is combined in a single object for the whole testing procedure.

3.1. Data, independence problems and test statistics

Data structure

We are provided with n observations $(\mathbf{Y}_i, \mathbf{X}_i, b_i, w_i), i = 1, \ldots, n$. In addition to variables $\mathbf{X}$, $\mathbf{Y}$, and b, it is convenient (for example to efficiently represent large contingency tables) to include case weights w_i, defaulting to $w_i \equiv 1$. This data structure is represented by class '`IndependenceProblem`':

Class '`IndependenceProblem`'

Slot	Class
`x`	'`data.frame`'
`y`	'`data.frame`'
`block`	'`factor`'
`weights`	'`numeric`'

Note that objects of this class implicitly define the null distribution H_0 and all admissible permutations of observations within blocks.

For our illustrating rotating rats example, we specify the hypothesis of independence of variables `time` and `group` by initializing a new independence problem with the corresponding observations:

```
R> ip <- new("IndependenceProblem",
+    y = rotarod["time"], x = rotarod["group"])
```

Independence problems and linear statistics

The transformation functions g and h as well as the transformed observations $g(\mathbf{X}_i)$ and $h(\mathbf{Y}_i), i = 1, \ldots, n$, are added to the data structure by extending class '`IndependenceProblem`':

Class '`IndependenceTestProblem`'
Contains '`IndependenceProblem`'

Slot	Class
`xtrans`	'`matrix`'
`ytrans`	'`matrix`'
`xtrafo`	'`function`'
`ytrafo`	'`function`'

The `ytrafo` and `xtrafo` slots correspond to the transformations h and g, respectively. The ith row of the $n \times q$ matrix `ytrans` corresponds to $h(\mathbf{Y}_i)$. Similarly, the rows of `xtrans` $(n \times p)$ correspond to $g(\mathbf{X}_i)$. Note that, in addition to the data, hypothesis and permutation scheme, the test statistic $\mathbf{T}$ is defined by objects of class '`IndependenceTestProblem`' as well.

In the simplest case of both $\mathbf{X}$ and $\mathbf{Y}$ being univariate factors at p and q levels, g and h are the corresponding dummy codings and the linear statistic $\mathbf{T}$ is the (vectorized) $p \times q$ contingency table of $\mathbf{X}$ and $\mathbf{Y}$. In the rats example, the default dummy coding for factor `group` is employed and a rank transformation (via `rank()`) is applied to `time`.

```
R> itp <- new("IndependenceTestProblem", ip, ytrafo = rank)
```

The linear statistic $\mathbf{T}$, its conditional expectation μ and covariance Σ are stored in objects of class 'IndependenceLinearStatistic':

Class 'IndependenceLinearStatistic'
Contains 'IndependenceTestProblem'

Slot	Class
linearstatistic	'numeric'
expectation	'numeric'
covariance	'VarCovar'

Class 'VarCovar' represents either a complete covariance matrix or its diagonal elements only. By default, only the conditional variances are stored, the whole covariance matrix can be computed as needed (see below). In the rotating rats example, such an object is easily created via

```
R> ils <- new("IndependenceLinearStatistic", itp)
```

Using methods for suitable generic functions (see Table 2), the linear statistic for the rotating rats can be extracted via

```
R> statistic(ils, "linear")

control 180
```

This is simply the sum of the average ranks in the control group, i.e., $180 = 12 \cdot 15$ because all 12 observations have the maximal average rank 15. Additionally, the associated conditional mean and variance under H_0 can be computed via:

```
R> expectation(ils)

control
    150

R> variance(ils)

 control
151.3043
```

based upon which we can now set up a test statistic.

Test statistics

The specification of the inference procedure is completed by the definition of a univariate test statistic c which is represented by a virtual class 'IndependenceTestStatistic'

Function	Description
`statistic(object, type)`	Extraction of the linear statistic **t** (`type = "linear"`), the standardized statistic **z** (`type = "standardized"`) or the final test statistic c (`type = "test"`, only for objects inheriting from '`IndependenceTestStatistic`').
`expectation(object)`	Extraction of the conditional expectation μ.
`covariance(object)`	Extraction of the complete conditional covariance matrix Σ.
`variance(object)`	Extraction of the diagonal elements of the conditional covariance matrix $\text{diag}(\Sigma)$.

Table 2: List of generic functions with methods for classes inheriting from '`IndependenceLinearStatistic`'.

Class '`IndependenceTestStatistic`'
Contains '`IndependenceLinearStatistic`'

Slot	Class
`teststatistic`	'`numeric`'
`standardizedlinearstatistic`	'`numeric`'

The slot `standardizedlinearstatistic` contains **z**, the (possibly multivariate) linear statistic standardized by its conditional expectation and variance (Equation 2). Slot `teststatistic` is for storing univariate test statistics c?

```
R> its <- new("IndependenceTestStatistic", ils)
```

Methods for all generics listed in Table 2 are also available for objects of this class.

The slot `teststatistic` has not yet been filled in the '`IndependenceTestStatistic`' object. **coin** implements three sub-classes with associated univariate test statistics for this: scalar test statistics c_{scalar}, maximum-type statistics $c_{\max}$ and quadratic forms c_{quad}. In case of univariate linear statistics **t**, i.e., for $pq = 1$, a natural test statistic c is simply the standardized linear statistic

$$c_{\text{scalar}}(\mathbf{t}, \mu, \Sigma) = \frac{\mathbf{t} - \mu}{\sqrt{\Sigma}} = \mathbf{z}.$$

A special class is available for this

Class '`ScalarIndependenceTestStatistic`'
Contains '`IndependenceTestStatistic`'

Slot	Class
`alternative`	'`character`'

that also defines a character vector specifying the alternative to test against (`"two.sided"`, `"greater"` and `"less"`). Thus, the construction of a scalar test statistic corresponds to the construction of a suitable object via

```
R> sits <- new("ScalarIndependenceTestStatistic", its,
+    alternative = "two.sided")
R> statistic(sits, "standardized")

control 2.438909
```

which yields the standardized Wilcoxon-Mann-Whitney statistic reported in the introductory example. In the multivariate case ($pq > 1$), a natural extension is to employ a maximum-type statistic of the form

$$c_{\max}(\mathbf{t}, \mu, \Sigma) = \begin{cases} \max |\mathbf{z}| & (\text{“two-sided”}), \\ \min (\mathbf{z}) & (\text{“less”}), \\ \max (\mathbf{z}) & (\text{“greater”}), \end{cases}$$

where the definition reflects the associated alternative with name given in quotes. Again a special class ‘`MaxTypeIndependenceTestStatistic`’ is available for this:

Class ‘`MaxTypeIndependenceTestStatistic`’
Contains ‘`IndependenceTestStatistic`’

Slot	Class
`alternative`	‘`character`’

Alternatively, a quadratic form $c_{\text{quad}}(\mathbf{t}, \mu, \Sigma) = (\mathbf{t} - \mu)^\top \Sigma^+ (\mathbf{t} - \mu)$ can be used as test statistic. It is computationally more expensive because the Moore-Penrose inverse Σ^+ of Σ is involved. Such statistics are represented by objects of class ‘`QuadTypeIndependenceTestStatistic`’ defining slots for Σ^+ and its rank (degrees of freedom):

Class ‘`QuadTypeIndependenceTestStatistic`’
Contains ‘`IndependenceTestStatistic`’

Slot	Class
`covarianceplus`	‘`matrix`’
`df`	‘`numeric`’

A slot `alternative` is not needed because, by construction, quadratic forms cannot be applied to one-sided hypotheses.

3.2. Representation of conditional null distributions

The conditional distribution (or an approximation thereof) and thus the p value corresponding to the statistic $c(\mathbf{t}, \mu, \Sigma)$ can be computed in several different ways. For some special forms of the linear statistic, the exact distribution of the test statistic is tractable. For 2-sample problems, the shift algorithm by Streitberg and Röhmel (1986, 1987) and the split-up algorithm by van de Wiel (2001) are implemented as part of the package.

Conditional Monte-Carlo procedures can always be used to approximate the exact distribution. In this case, within each block, a sufficiently large number of random samples from all admissible permutations of the observations is drawn. The test statistic is computed for

the permuted $\mathbf{X}$ values and the distribution of these test statistics is an approximation to the conditional reference distribution. When p values are computed, confidence intervals are available from the binomial distribution.

Strasser and Weber (1999, Theorem 2.3) showed that the conditional distribution of linear statistics $\mathbf{T}$ with conditional expectation μ and covariance Σ tends to a multivariate normal distribution with parameters μ and Σ as $\sum_{i=1}^{n} I(b_i = j)w_i \to \infty$ for all $j = 1, \ldots, k$. Thus, the asymptotic conditional distribution of the standardized linear statistic $\mathbf{z}$ is normal and therefore, p values for scalar or maximum-type univariate statistics can be computed directly in the univariate case ($pq = 1$) or approximated by numerical algorithms (Genz 1992) as implemented in package **mvtnorm** (Genz, Bretz, and Hothorn 2008) in the multivariate setting. For quadratic forms c_{quad} which follow a χ^2 distribution with degrees of freedom given by the rank of Σ (see Johnson and Kotz 1970, Chapter 29), asymptotic probabilities can be computed straightforwardly.

A null distribution is represented by either a distribution (and p value) function only

Class '`PValue`'

Slot	Class
`pvalue`	'`function`'
`p`	'`function`'
`name`	'`character`'

or, where possible, is augmented by its density and quantile functions:

Class '`NullDistribution`'
Contains '`PValue`'

Slot	Class
`q`	'`function`'
`d`	'`function`'
`support`	'`function`'
`parameters`	'`list`'

Currently, there are three classes extending '`NullDistribution`' (without defining additional slots at the moment): '`ExactNullDistribution`', '`ApproxNullDistribution`', and '`AsymptNullDistribution`'. All of them can be queried for probabilities, quantiles, etc., using suitable methods (see Table 3 for an overview). New methods for computing or approximating the conditional distribution can be integrated into the framework via suitable inheritance from '`PValue`' (an example is given in Section 4).

The `support` function returns the support of a discrete distribution or an interval containing 99.999% of the probability mass.

Using these tools, the exact p value for the independence test on the rotating rats along with the complete exact reference distribution (allowing for the computation of quantiles, for example) can be derived via

```
R> end <- ExactNullDistribution(sits)
R> pvalue(end, statistic(sits))
```

Class	Description
'ExactNullDistribution'	Exact conditional null distribution (e.g., computed via the shift algorithm).
'ApproxNullDistribution'	Approximation of the exact conditional distribution using conditional Monte-Carlo procedures.
'AsymptNullDistribution'	Asymptotic conditional distribution (via multivariate normal or χ^2 distribution).
Method	**Description**
pvalue(object)	Computation of the p value (plus a confidence interval if Monte-Carlo procedures have been used) based on an observed test statistic c and its conditional null distribution.
pperm(object, q)	Evaluation of the cumulative distribution function for quantile q.
dperm(object, x)	Evaluation of the probability density function at x.
qperm(object, p)	Evaluation of the quantile function for probability p.
support(object)	Extraction of the support of the null distribution.

Table 3: Classes and methods for conditional null distributions.

```
[1] 0.03726708
```

```
R> qperm(end, 0.95)
```

```
[1] 1.544642
```

For maximum-type statistics $c_{\max}$, single-step and step-down multiplicity adjusted p values based on the limiting distribution and conditional Monte-Carlo methods (see Westfall and Young 1993) are available as well.

3.3. Objects for conditional tests

A conditional test is represented by a test statistic of class 'IndependenceTestStatistic' and its conditional null distribution inheriting from class 'PValue'. In addition, a character string giving the name of the test procedure is defined in class 'IndependenceTest':

Class 'IndependenceTest'

Slot	Class
distribution	'PValue'
statistic	'IndependenceTestStatistic'
estimates	'list'
method	'character'

Remember that objects of class 'IndependenceTestStatistic' represent the data, hypothesis, linear statistic and test statistic along with conditional expectation and covariance matrix.

The `estimates` slot may contain parameter estimates where available, for example an estimate and corresponding confidence interval for a shift parameter derived from a conditional Wilcoxon-Mann-Whitney test.

A complete description of the conditional independence test for the rotating rats data is given by an object of class '`IndependenceTest`' which is conveniently represented by the corresponding `show()` method:

```
R> new("IndependenceTest", statistic = sits, distribution = end)

        Exact General Independence Test

data:  time by group (control, treatment)
c = 2.4389, p-value = 0.03727
```

Of course, the methods previously defined in this section (see Tables 2 and 3) are defined for objects of class '`IndependenceTest`' as well. Thus, all theoretical entities introduced in Section 2 are now captured in a single object of class '`IndependenceTest`' and all methods for extracting information from it are readily available.

4. Interfaces to permutation inference

In Section 3, all the necessary computational building blocks are introduced for implementing the general class of permutation tests outlined in Section 2. However, one rarely needs to exploit the full flexibility of each component of the framework. More often, one wants to employ sensible defaults for most (if not all) steps in the analysis but preserving the possibility to extend a few steps based on user-supplied methods. For this purpose, **coin** provides the function `independence_test()` as the main user interface for performing independence tests. Many of its arguments have flexible defaults or can be specified by a simple character string while still allowing to plug in much more complex user-defined objects, e.g., for data preparation, computation of the null distribution, or transformation functions.

4.1. A convenient user interface

Via the generic function `independence_test()`, all steps described in Section 3 can be carried out using a single command. The main workhorse behind it is the method for objects of class '`IndependenceProblem`':

```
independence_test(object,
  teststat = c("max", "quad", "scalar"),
  distribution = c("asymptotic", "approximate", "exact"),
  alternative = c("two.sided", "less", "greater"),
  xtrafo = trafo, ytrafo = trafo, scores = NULL,
  check = NULL, ...)
```

Thus, `object` describes the data and the null hypothesis. Arguments `xtrafo` and `ytrafo` refer to the transformations g and h: Both are by default set to function `trafo()` which chooses suitable transformations based on the scale of the considered variables (see below). The three

types of univariate test statistics discussed above are hard-coded and can be specified by a simple string. Similarly, the reference distribution and the alternative hypothesis can be supplied as strings. In addition to these simple specifications, more flexible specifications for the data (`object`), the transformations (`xtrafo`, `ytrafo`), and the distribution are available, as discussed in the following. The `scores` argument takes a named list of numeric vectors to be used as scores for ordered factors. Validity checks for objects of class '`IndependenceProblem`' specified to argument `check` can be used to test for certain aspects of the data, e.g., when one has to make sure that two independent samples are present.

4.2. Data specification

The standard way of specifying relationships between variables in R are formulas in combination with a data frame. Hence, a '`formula`' method for `independence_test()` is provided that interprets the left hand side variables of a formula as **Y** variables (univariate or possibly multivariate), the right hand side as **X** variables (univariate or multivariate as well). An optional blocking factor can specified after a vertical bar, e.g.,

```
y1 + y2 ~ x1 + x2 | block
```

This specifies an independence problem between two **Y** variables and two **X** variables (in case all variables are numeric the linear statistic is 4-dimensional with $p = 2$ and $q = 2$) for each level in `block`. As usual, `data`, `weights` and `subset` arguments can be specified as well. Based on all these arguments the '`IndependenceProblem`' is built and simply passed on to the `independence_test()` method described above.

For (simple) categorical data, there is yet another way of specifying the independence problem, namely via the '`table`' method of `independence_test()`. Its first argument is allowed to be a 2- or 3-dimensional table: The first two margins are interpreted as univariate categorical **X** and **Y** variables and the optional third margin is taken to be the blocking factor. Again, an '`IndependenceProblem`' object is derived from this and passed on to the associated method.

4.3. Null distributions

In the simplest case, the `distribution` argument of the `independence_test()` methods can be specified by a simple character string. However, this is not flexible enough in some situations and hence it is also possible to supply a function that can compute a '`PValue`' object from an '`IndependenceTestStatistic`' object.

For the most important special cases, suitable function *generators* are provided in **coin**. For example, the function `approximate(B = 1000)` returns a Monte Carlo function that draws `B` (default: 1000) random permutations. Similarly, `exact()` and `approximate()` return functions computing the exact or asymptotic null distributions, respectively. Again, computational details in the computation of the null distribution can be controlled via arguments of the function generators.

Additionally, it is also possible to set `distribution` to a user-supplied algorithm for computing the conditional null distribution. It just has to be provided in the form of a function taking an object inheriting from '`IndependenceTestStatistic`' and returning an object inheriting from class '`PValue`'.

As an example, consider the computation of the exact p value for testing independence of two continuous random variables. The identity transformation is used for both g and h, thus a conditional version of the test for zero Pearson correlation is constructed. We use a tiny artificial example where enumeration and evaluation of all possible permutations is still feasible. The function `sexact()` extracts both variables, computes all permutations using the function `permutations()` from **e1071** (Dimitriadou, Hornik, Leisch, Meyer, and Weingessel 2008), computes the linear test statistic for each permutation, standardizes it by expectation and variance, and then sets up the distribution function.

```
R> set.seed(2908)
R> correxample <- data.frame(x = rnorm(7), y = rnorm(7))
R> sexact <- function(object) {
+    x <- object@xtrans
+    y <- object@ytrans
+    perms <- permutations(nrow(x))
+    pstats <- apply(perms, 1, function(p) sum(x[p,] * y))
+    pstats <- (pstats - expectation(object)) / sqrt(variance(object))
+    p <- function(q) 1 - mean(pstats > q)
+    new("PValue", p = p, pvalue = p)
+  }
```

Note that above implementation is kept simple for the purpose of illustration; it hard-codes the alternative (less) and assumes that the transformed variables are univariate.

This function can then be passed to `independence_test()` for computing the distribution function and p value:

```
R> independence_test(y ~ x, data = correxample, alternative = "less",
+    distribution = sexact)

        General Independence Test

data:  y by x
Z = 1.4203, p-value = 0.9226
alternative hypothesis: less
```

4.4. Transformations

By default, `independence_test()` chooses the transformations g and h via the wrapper function `trafo()`. This, in turn, chooses the actual transformation based on the scale of each variable (individually) in a data frame `data`:

```
trafo(data, numeric_trafo = id_trafo, factor_trafo = f_trafo,
  surv_trafo = logrank_trafo, var_trafo = NULL, block = NULL)
```

The identity transformation is used for numeric variables (`id_trafo()`), factors are transformed to indicator variables (`f_trafo()`), and censored variables are transformed to log

rank scores (`logrank_trafo()`). In `f_trafo()` a set of k indicator functions is used for a factor with k levels, unless $k = 2$ for which a univariate indicator function is employed. A named list containing different transformations to be applied to certain variables may be specified as `var_trafo` argument. When a factor is given as `block`, all transformations are applied separately within each block. The function `trafo()` can also easily be re-used by supplying other (possibly user defined functions) as arguments to `trafo()`.

Instead of using `trafo()`, a user-defined transformation can also be passed directly to `xtrafo` or `ytrafo`. As an example, consider using a Mood test against scale alternatives for the rotating rats data. This amounts to using the transformation $h(Y_i) = (\text{rank}(Y_i) - (n+1)/2)^2$ for the response:

```
R> mood_score <- function(y) (rank(y) - (nrow(y) + 1) / 2)^2
```

This can be used for constructing an exact test (based on the split-up algorithm) by hand:

```
R> ip <- new("IndependenceProblem",
+    y = rotarod["time"], x = rotarod["group"])
R> itp <- new("IndependenceTestProblem", ip,
+    ytrafo = mood_score)
R> its <- new("IndependenceTestStatistic", itp)
R> sits <- new("ScalarIndependenceTestStatistic", its,
+    alternative = "two.sided")
R> new("ScalarIndependenceTest", statistic = sits,
+    distribution = ExactNullDistribution(sits, algorithm = "split-up"))

        Exact General Independence Test

data:  time by group (control, treatment)
Z = -2.3208, p-value = 0.03727
alternative hypothesis: two.sided
```

Alternatively, and more easily, the same result can be obtained by plugging `mood_score()` into `independence_test()`:

```
R> independence_test(time ~ group, data = rotarod, ytrafo = mood_score,
+    distribution = exact(algorithm = "split-up"))

        Exact General Independence Test

data:  time by group (control, treatment)
Z = -2.3208, p-value = 0.03727
alternative hypothesis: two.sided
```

Similarly to the Mood test, the conditional counterpart of many other classical tests (some of them implemented in package **stats**) are easily available through `independence_test()` by specifying the appropriate `xtrafo`, `ytrafo` and `teststat`. This includes the Wilcoxon-Mann-Whitney or Cochran-Mantel-Haenszel tests, but also many other well-known tests as shown in Table 4.

Test	`xtrafo` g	`ytrafo` h	`teststat` c
	Independent samples		
Wilcoxon-Mann-Whitney	`f_trafo()`	`rank()`	`"scalar"`
Normal quantiles	`f_trafo()`	`normal_trafo()`	`"scalar"`
Median	`f_trafo()`	`median_trafo()`	`"scalar"`
Ansari-Bradley	`f_trafo()`	`ansari_trafo()`	`"scalar"`
Log rank	`f_trafo()`	`logrank_trafo()`	`"quad"`
Kruskal-Wallis	`f_trafo()`	`rank()`	`"quad"`
Fligner	`f_trafo()`	`fligner_trafo()`	`"quad"`
Spearman	`rank()`	`rank()`	`"scalar"`
Cochran-Mantel-Haenszel	`f_trafo()`	`f_trafo()`	`"quad"`
Pearson's χ^2	`f_trafo()`	`f_trafo()`	`"quad"`
Cochran-Armitage / Linear Association	scores	any	`"scalar"`
K-sample permutation test	`f_trafo()`	any	any
Maximally-selected statistics	`maxstat_trafo()`	any	`"max"`
	Dependent samples		
Friedman	`f_trafo()`	`rank()`	`"quad"`
Maxwell-Stuart	`f_trafo()`	`f_trafo()`	`"quad"`
Wilcoxon signed rank	`f_trafo()`	`rank()`	`"scalar"`

Table 4: Representations of the conditional counterparts of important classical tests in **coin**.

Due to this flexibility, almost all special-purpose functionality implemented in packages **exactRankTests** (Hothorn 2001; Hothorn and Hornik 2002, 2006) and **maxstat** (Hothorn and Lausen 2002; Hothorn 2007) can be conveniently provided within the **coin** framework, so that both packages will become deprecated in the future.

5. Permutation tests in practice: A categorical data example

The job satisfaction of African-American males in the USA (Agresti 2002, Table 7.8) is described by measures of income and reported job satisfaction in four (ordinal) classifications. It seems natural to surmise that job satisfaction increases with income. The data (see Figure 1) is given as a three-dimensional '`table`' with variables `Income` and `Job.Satisfaction` according to `Gender` (labels slightly modified for convenience):

```
R> data("jobsatisfaction", package = "coin")
R> js <- jobsatisfaction
R> dimnames(js)[[2]] <- c("VeryDiss", "LitSat", "ModSat", "VerySat")
R> ftable(Job.Satisfaction ~ Gender + Income, data = js)
```

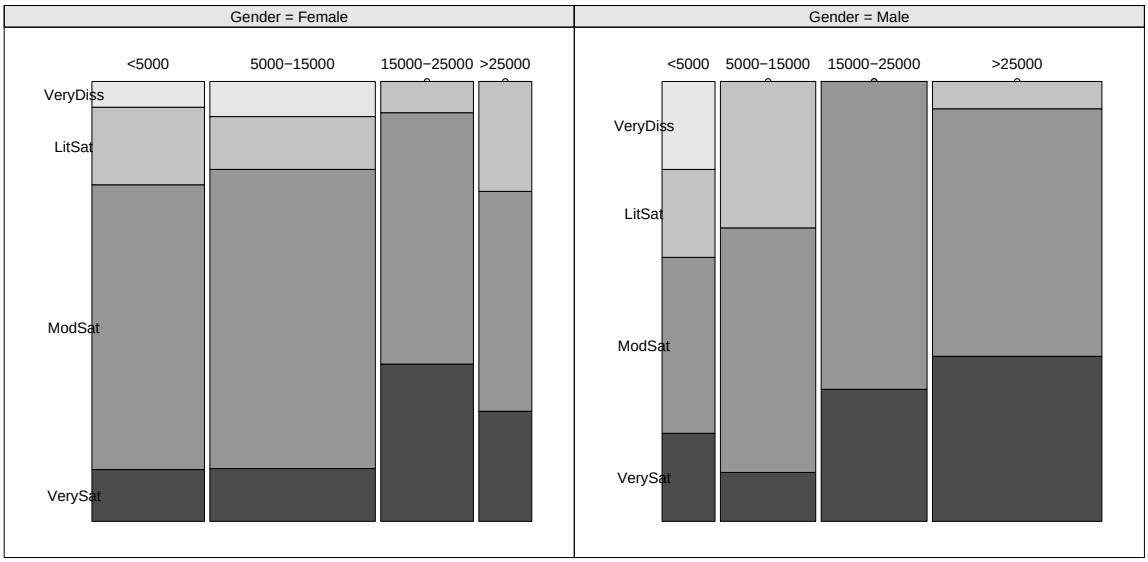

Figure 1: Conditional mosaic plot of job satisfaction and income given gender.

```
                Job.Satisfaction VeryDiss LitSat ModSat VerySat
Gender Income
Female <5000                           1       3     11       2
       5000-15000                      2       3     17       3
       15000-25000                     0       1      8       5
       >25000                          0       2      4       2
Male   <5000                           1       1      2       1
       5000-15000                      0       3      5       1
       15000-25000                     0       0      7       3
       >25000                          0       1      9       6
```

The Cochran-Mantel-Haenszel test—a classical test for testing independence in stratified contingency tables—could be used for assessing the independence hypothesis of income and job satisfaction (stratified by gender). Traditionally, this test employs a c_{quad} statistic derived from the contingency table and a χ^2 approximation of the null distribution:

```
R> it <- independence_test(js, teststat = "quad",
+    distribution = asymptotic())
R> it

        Asymptotic General Independence Test

data:  Job.Satisfaction by
         Income (<5000, 5000-15000, 15000-25000, >25000)
         stratified by Gender
chi-squared = 10.2001, df = 9, p-value = 0.3345
```

Thus, the test does not indicate significant departure from independence. However, ordering of the factors is not exploited by the Cochran-Mantel-Haenszel test. As some positive correlation of the two factors would seem natural, it is worth having a closer look at the data and the test result. The underlying linear statistic is

```
R> statistic(it, "linear")

            VeryDiss LitSat ModSat VerySat
<5000               2      4     13       3
5000-15000          2      6     22       4
15000-25000         0      1     15       8
>25000              0      3     13       8
```

This is exactly the original contingency table aggregated over the block factor `Gender`:

```
R> margin.table(js, 1:2)

             Job.Satisfaction
Income        VeryDiss LitSat ModSat VerySat
  <5000               2      4     13       3
  5000-15000          2      6     22       4
  15000-25000         0      1     15       8
  >25000              0      3     13       8
```

Therefore, the standardized linear statistic can be interpreted similarly to Pearson residuals for the independence hypothesis:

```
R> statistic(it, "standardized")

              VeryDiss      LitSat     ModSat    VerySat
<5000        1.3112789  0.69201053 -0.2478705 -0.9293458
5000-15000   0.6481783  0.83462550  0.5175755 -1.6257547
15000-25000 -1.0958361 -1.50130926  0.2361231  1.4614123
>25000      -1.0377629 -0.08983052 -0.5946119  1.2031648
```

The positive diagonal and (mostly) negative off-diagonal elements convey that higher income categories seem to be associated with higher job satisfaction. Thus, to direct power against ordered alternatives, a linear-by-linear association statistic (Agresti 2002) should be used instead of the omnibus χ^2 statistic. This can be conveniently performed within **coin**, e.g., using simple equi-distant scores and a Monte Carlo approximation of the null distribution:

```
R> it <- independence_test(js, distribution = approximate(B = 10000),
+    scores = list(Job.Satisfaction = 1:4, Income = 1:4))
R> pvalue(it)

[1] 0.0116
99 percent confidence interval:
 0.009024885 0.014651019
```

Using this strategy, the null hypothesis of independence of job satisfaction and income can be rejected in favor of a positive association of both variables. Other choices of scores are also conceivable. Especially when there is an underlying numeric scale, interval midpoints are often used (see Hothorn *et al.* 2006, for an example).

For other patterns of dependence—e.g., when only a few cells in a large table deviate from independence—a maximum-type statistic is also useful for contingency tables. To complete our tour of **coin** tools for categorical data, we briefly illustrate this approach using the job satisfaction data again (even though a maximum-type statistic is clearly not very powerful for the dependence pattern in this data set). The maximum-type test is set up easily:

```
R> independence_test(js, teststat = "max")

        Asymptotic General Independence Test

data:  Job.Satisfaction by
         Income (<5000, 5000-15000, 15000-25000, >25000)
         stratified by Gender
maxT = 1.6258, p-value = 0.7214
```

with its conditional asymptotic null distribution being available immediately (due to the joint multivariate normal distribution for the contingency table $\mathbf{T}$). Single-step adjusted p values for each cell of the contingency table corresponding to this maximum test can be computed via

```
R> pvalue(independence_test(js, teststat = "max"), method = "single-step")

             VeryDiss    LitSat    ModSat   VerySat
<5000       0.9010585 0.9987783 0.9999998 0.9888276
5000-15000  0.9992724 0.9948608 0.9998850 0.7215165
15000-25000 0.9660054 0.8035550 0.9999999 0.8269899
>25000      0.9761862 1.0000000 0.9996381 0.9394694
```

These p values can be interpreted in a way similar to standardized contingency tables. The discrepancy between the global adjusted p value shown above and the minimum single-step adjusted p value is due to simulation variance. For more practical examples, including applications with numeric variables, we refer to Hothorn *et al.* (2006).

6. Odds and ends

Internal functionality. The core functionality, i.e., a small set of functions computing the linear statistic $\mathbf{T}$ (both for the original and permuted data), the conditional expectation μ and conditional covariance matrix Σ, is coded in C. The shift and split-up algorithms (Streitberg and Röhmel 1986, 1987; van de Wiel 2001) for computing the exact null distribution in 2-sample problems with univariate response as well as conditional Monte-Carlo procedures for approximating the exact conditional null distribution are implemented in C as well. (In addition, some helper functions, e.g., the Kronecker product etc., are coded in C.) The complete C source code and its documentation can be accessed via

```
R> browseURL(system.file("documentation", "html", "index.html",
+    package = "coin"))
```

The naming scheme of the C routines distinguishes between functions only called at the C level (`C_`*foo*) and functions which can be called from R via the `.Call` interface (`R_`*foo*). Such functions are available for most of the internal C functions to enable unit testing.

Quality assurance. The test procedures implemented in **coin** are continuously checked against results obtained by the corresponding implementations in **stats** (where available). In addition, the test statistics and exact, approximate and asymptotic p values for data examples given in the **StatXact** 6 user manual (Cytel Inc. 2003) are compared with the results reported there. Step-down multiple adjusted p values have been checked against results reported by `mt.maxT()` from package **multtest** (Pollard, Ge, and Dudoit 2008). For details on the test procedures we refer to the R transcript files in directory '`tests`' of the **coin** package sources.

Computational details. The **coin** package imports packages **mvtnorm** (Genz *et al.* 2008) for the evaluation of the multivariate normal distribution and package **modeltools** (Hothorn, Leisch, and Zeileis 2008b) for formula parsing. The class structure, internal functionality, user interface and examples are based on **coin** version 1.0-3, available under the terms of the General Public License from `http://CRAN.R-project.org/`. R version 2.8.1 (R Development Core Team 2008) was used for the computations, Figure 1 was created using the **vcd** package (Meyer, Zeileis, and Hornik 2006).

Acknowledgments

We would like to thank Helmut Strasser for discussions on the theoretical framework. Henric Nilsson provided clarification and examples for the Maxwell-Stuart test and helped identifying bugs. The work of Torsten Hothorn was supported by Deutsche Forschungsgemeinschaft (DFG) under grant HO 3242/1-3.

References

Agresti A (2002). *Categorical Data Analysis*. John Wiley & Sons, Hoboken, New Jersey, 2nd edition.

Bergmann R, Ludbrook J, Spooren WPJM (2000). "Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages." *The American Statistician*, **54**(1), 72–77.

Cytel Inc (2003). ***StatXact*** *6: Statistical Software for Exact Nonparametric Inference*. Cytel Software Corporation, Cambridge, MA. URL `http://www.cytel.com/`.

Cytel Inc (2006). ***LogXact*** *8: Discrete Regression Software Featuring Exact Methods*. Cytel Software Corporation, Cambridge, MA. URL `http://www.cytel.com/`.

Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A (2008). ***e1071**: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.5-18, URL http://CRAN.R-project.org/package=e1071.

Ernst MD (2004). "Permutation Methods: A Basis for Exact Inference." *Statistical Science*, **19**(4), 676–685.

Fisher RA (1935). *The Design of Experiments*. Oliver and Boyd, Edinburgh, UK.

Genz A (1992). "Numerical Computation of Multivariate Normal Probabilities." *Journal of Computational and Graphical Statistics*, **1**, 141–149.

Genz A, Bretz F, Hothorn T (2008). ***mvtnorm**: Multivariate Normal and T Distribution*. R package version 0.9-2, URL http://CRAN.R-project.org/package=mvtnorm.

Hájek J, Šidák Z, Sen PK (1999). *Theory of Rank Tests*. Academic Press, London, 2nd edition.

Hothorn T (2001). "On Exact Rank Tests in R." *R News*, **1**(1), 11–12. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/.

Hothorn T (2007). ***maxstat**: Maximally Selected Rank Statistics*. R package version 0.7-12, URL http://CRAN.R-project.org/package=maxstat.

Hothorn T, Hornik K (2002). "Exact Nonparametric Inference in R." In W Härdle, B Rönz (eds.), "Proceedings in Computational Statistics: COMPSTAT 2002," pp. 355–360. Physica-Verlag, Heidelberg.

Hothorn T, Hornik K (2006). ***exactRankTests**: Exact Distributions for Rank and Permutation Tests*. R package version 0.8-17, URL http://CRAN.R-project.org/package=exactRankTests.

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2006). "A Lego System for Conditional Inference." *The American Statistician*, **60**(3), 257–263. doi:10.1198/000313006X118430.

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2008a). "Implementing a Class of Permutation Tests: The **coin** Package." *Journal of Statistical Software*, **28**(8), 1–23. URL http://www.jstatsoft.org/v28/i08/.

Hothorn T, Lausen B (2002). "Maximally Selected Rank Statistics in R." *R News*, **2**(1), 3–5. ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/.

Hothorn T, Leisch F, Zeileis A (2008b). ***modeltools**: Tools and Classes for Statistical Models*. R package version 0.2-16, URL http://CRAN.R-project.org/package=modeltools.

Johnson NL, Kotz S (1970). *Distributions in Statistics: Continuous Univariate Distributions 2*. John Wiley & Sons, New York.

Meyer D, Zeileis A, Hornik K (2006). "The Strucplot Framework: Visualizing Multi-Way Contingency Tables with **vcd**." *Journal of Statistical Software*, **17**(3). URL http://www.jstatsoft.org/v17/i03/.

Oster RA (2002). "An Examination of Statistical Software Packages for Categorical Data Analysis Using Exact Methods." *The American Statistician*, **56**(3), 235–246.

Oster RA (2003). "An Examination of Statistical Software Packages for Categorical Data Analysis Using Exact Methods–Part II." *The American Statistician*, **57**(3), 201–213.

Pollard KS, Ge Y, Dudoit S (2008). ***multtest**: Resampling-Based Multiple Hypothesis Testing*. R package version 1.21.1, URL http://CRAN.R-project.org/package=multtest.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

SAS Institute Inc (2003). *SAS/STAT Software, Version 9.1*. Cary, NC. URL http://www.sas.com/.

StataCorp (2003). *Stata Statistical Software: Release 8*. StataCorp LP, College Station, TX. URL http://www.stata.com/.

Strasser H, Weber C (1999). "On the Asymptotic Theory of Permutation Statistics." *Mathematical Methods of Statistics*, **8**, 220–250. Preprint available from http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01_94c.

Streitberg B, Röhmel J (1986). "Exact Distributions for Permutation and Rank Tests: An Introduction to Some Recently Published Algorithms." *Statistical Software Newsletter*, **12**(1), 10–17. ISSN 1609-3631.

Streitberg B, Röhmel J (1987). "Exakte Verteilungen für Rang- und Randomisierungstests im allgemeinen c-Stichprobenfall." *EDV in Medizin und Biologie*, **18**(1), 12–19.

van de Wiel MA (2001). "The Split-Up Algorithm: A Fast Symbolic Method for Computing p Values of Rank Statistics." *Computational Statistics*, **16**, 519–538.

Westfall PH, Young SS (1993). *Resampling-Based Multiple Testing*. John Wiley & Sons, New York.

A. Expectation and covariance

The conditional expectation and covariance matrix of linear statistics $\mathbf{T}$ as given in Equation 1 in Section 2 are computed as follows. Let $w_{\cdot j} = \sum_{i=1}^{n} I(b_i = j) w_i$ denote the sum of the weights in block j and S_j the set of all permutations of the observations in block j. The conditional expectation of the transformation h in block j is

$$\mathsf{E}(h|S_j) = w_{\cdot j}^{-1} \sum_i I(b_i = j) w_i h(\mathbf{Y}_i) \in \mathbb{R}^q$$

with corresponding $q \times q$ covariance matrix

$$\mathsf{COV}(h|S_j) = w_{\cdot j}^{-1} \sum_i I(b_i = j) w_i \left(h(\mathbf{Y}_i) - \mathsf{E}(h|S_j)\right) \left(h(\mathbf{Y}_i) - \mathsf{E}(h|S_j)\right)^\top.$$

This is the basis for computing the conditional expectation and covariance of the linear statistic $\mathbf{T}_j$ in block j:

$$\mu_j \quad = \quad \mathsf{E}(\mathbf{T}_j|S_j) = \mathrm{vec}\left(\left(\sum_{i=1}^{n} I(b_i = j) w_i g(\mathbf{X}_i)\right) \mathsf{E}(h|S_j)^\top\right)$$

and

$$\begin{aligned}
\Sigma_j &= \mathsf{COV}(\mathbf{T}_j|S_j) \\
&= \frac{w_{\cdot j}}{w_{\cdot j} - 1} \mathsf{COV}(h|S_j) \otimes \left(\sum_i I(b_i = j) w_i \left(g(\mathbf{X}_i) \otimes g(\mathbf{X}_i)^\top\right)\right) \\
&- \frac{1}{w_{\cdot j} - 1} \mathsf{COV}(h|S_j) \otimes \left(\sum_i I(b_i = j) w_i g(\mathbf{X}_i)\right) \otimes \left(\sum_i I(b_i = j) w_i g(\mathbf{X}_i)\right)^\top
\end{aligned}$$

where $\otimes$ is the Kronecker product. The conditional expectation and covariance of $\mathbf{T}$, aggregated over all k blocks, are then given by

$$\begin{aligned}
\mathsf{E}(\mathbf{T}|S_j) \quad = \quad \mu \quad &= \quad \sum_{j=1}^{k} \mu_j \quad = \quad \sum_{j=1}^{k} \mathsf{E}(\mathbf{T}_j|S_j), \\
\mathsf{COV}(\mathbf{T}|S_j) \quad = \quad \Sigma \quad &= \quad \sum_{j=1}^{k} \Sigma_j \quad = \quad \sum_{j=1}^{k} \mathsf{COV}(\mathbf{T}_j|S_j).
\end{aligned}$$

Affiliation:

Torsten Hothorn
Institut für Statistik
Ludwig-Maximilians-Universität München
Ludwigstraße 33
DE-80539 München, Germany
E-mail: Torsten.Hothorn@R-project.org
URL: http://www.stat.uni-muenchen.de/~hothorn/

Kurt Hornik, Achim Zeileis
Department of Statistics and Mathematics
Wirtschaftsuniversität Wien
Augasse 2–6
AT-1090 Wien, Austria
E-mail: Kurt.Hornik@R-project.org, Achim.Zeileis@R-project.org

Mark A. van de Wiel
Department of Mathematics
Vrije Universiteit Amsterdam
De Boelelaan 1081a
NL-1081 HV Amsterdam, The Netherlands
E-mail: mark.vdwiel@vumc.nl