μTOSS - Multiple hypothesis testing in an open software system

Gilles Blanchard GILLES.BLANCHARD@WIAS-BERLIN.DE
Weierstrass Institute for Applied Analysis and Stochastics Berlin

Thorsten Dickhaus DICKHAUS@MATH.HU-BERLIN.DE
Humboldt-University Berlin

Niklas Hack NIKLAS.HACK@MEDUNIWIEN.AC.AT
Medical University of Vienna

Frank Konietschke FRANK.KONIETSCHKE@MEDIZIN.UNI-GOETTINGEN.DE
Georg-August-University Göttingen

Kornelius Rohmeyer ROHMEYER@BIOSTAT.UNI-HANNOVER.DE
Leibniz University Hannover

Jonathan Rosenblatt JOHN.ROS@GMAIL.COM
Tel Aviv University

Marsel Scheer MSCHEER@DDZ.UNI-DUESSELDORF.DE
German Diabetes Center

Wiebke Werft W.WERFT@DKFZ-HEIDELBERG.DE
German Cancer Research Center

Editors: Tom Diethe, Nello Cristianini, John Shawe-Taylor

Abstract

μTOSS is an `R` package providing an open source, easy-to-extend platform for multiple hypothesis testing (MHT), one of the most active research fields in statistics over the last 10-15 years. Its first motivation is to establish a common platform and standardization for MHT procedures at large. The μTOSS software has been designed and written in the framework of a "Harvest Programme" call of the PASCAL2 European research network. Basically, it consists of the two R packages `mutoss` and `mutossGUI`. For researchers, it features a convenient unification of interfaces for MHT procedures (including standardized functions to access existing specific MHT `R` packages such as `multtest` and `multcomp`, as well as recent MHT procedures that are not available elsewhere) and helper functions facilitating the setup of benchmark simulations for comparison of competing methods. For end users, a graphical user interface and an online user's guide for finding appropriate methods for a given specification of the multiple testing problem is included. Ongoing maintenance and subsequent extensions will aim at establishing μTOSS as a state of the art in statistical computing for MHT.

1. Introduction

1.1. Motivation

Multiple hypothesis testing (MHT) has emerged as one of the most active research fields in statistics over the last 10-15 years, especially driven by large-scale applications, such as genomics, proteomics or cosmology. According to data presented by Benjamini (2009), approximately 8% of all articles in the four leading methodological statistical journals nowadays deal with multiple testing. Many new multiple (type I and type II) error criteria like the meanwhile quite popular "false discovery rate" (FDR) have recently been propagated and published together with explicit algorithms for controlling them. Moreover, regulatory agencies have also become more and more aware of multiplicity and reproducibility issues in research and therefore require multiplicity-adjusted confidence regions for reported effects whereever possible (cf., e. g., U. S. Food and Drug Administration, 2010).

One major characteristic of recent research in this field consists in more or less direct implementation of the theoretical results into (individual) software. It is fair to say that up to now every research group uses its own software, making (simulation) study evaluations and related results not entirely comparable. Moreover, the spread of newly emerging methods is hindered by the lack of a common software platform to agree on.

Motivated by a suggestion made by Benjamini (2009), we developed an open software framework for multiple hypotheses testing called "μTOSS", sponsored by the PASCAL2 European Network of Excellence and realized at Berlin Institute of Technology. It is based on the freely available statistical computing environment `R`, see `http://www.r-project.org/`.

The μTOSS software is targeted towards researchers working theoretically in the field of multiple hypothesis testing as well as to end users working in the life sciences. For the first target group, the unification of interfaces for many known MHT procedures, should in particular prove useful. Moreover, the simulation tool described in Section 3 facilitates the comparison of competing methods. End users will profit from the graphical user interface (GUI, cf. Section 4) and its filter mechanism. The latter serves as a guide to appropriate procedures depending on the user's specification of the test problem.

1.2. Multiple hypothesis testing in a nutshell

Let a statistical model $(\mathcal{X}, \mathcal{A}, (\mathbb{P}_\vartheta)_{\vartheta \in \Theta})$ parametrized by $\vartheta \in \Theta$ be given. Multiple hypothesis testing is concerned with testing a family $\mathcal{H} = (H_i, i \in I)$ of hypotheses regarding the parameter ϑ with corresponding alternatives $K_i = \Theta \setminus H_i$. A multiple test procedure $\varphi = (\varphi_i, i \in I)$ is a sequence of tests. Its components φ_i map from $\mathcal{X}$ into $\{0, 1\}$ and have the usual interpretation of a statistical test for the pair of hypotheses H_i versus K_i, namely, $\varphi_i(x) = 1$ means rejection of H_i and decision in favor of K_i whereas $\varphi_i(x) = 0$ means no rejection. Let $I_0 \equiv I_0(\vartheta) \subseteq I$ denote the index set of true hypotheses in $\mathcal{H}$. Moreover, let $R(\varphi)$ denote the total number of rejections and $V(\varphi)$ the number of type I errors of φ, i. e., $R(\varphi) = \sum_{i \in I} \varphi_i$ and $V(\varphi) = \sum_{i \in I_0} \varphi_i$. The classical multiple type I error measure is the family-wise error rate (FWER) and can be expressed as $\mathrm{FWER}_\vartheta(\varphi) = \mathbb{P}_\vartheta(V(\varphi) > 0)$. In words, $\mathrm{FWER}_\vartheta(\varphi)$ denotes the probability (under $\vartheta \in \Theta$) that φ leads to at least one type I error. For multiple testing problems of massive size ($|\mathcal{H}|$ very large), the false discovery rate (FDR), defined as the expected proportion of type I errors among all rejections (in formula:

$\text{FDR}_\vartheta(\varphi) = \mathbb{E}_\vartheta[V(\varphi)/\max(R(\varphi), 1)])$ has become a very popular alternative type I error measure during the last 15 years. A vast diversity of alternative criteria and corresponding type II analogues has been developed.

It goes beyond the scope of this paper to describe techniques for controlling the FWER or the FDR (control of an error rate thereby means finding a multiple test φ guaranteeing that the error rate is bounded from above by a fixed significance level $\alpha \in (0, 1)$ for all possible parameter values $\vartheta \in \Theta$). However, it is worth mentioning that many closely related multiple statistical decision problems can be solved with MHT techniques, e. g., constructing simultaneous confidence regions for model parameters or selection, partitioning and ranking problems. Therefore, μTOSS provides functions for computing simultaneous confidence intervals for parameters of specific models in addition to the mere determination of rejection patterns.

2. General design paradigms and overview of methods

2.1. General objectives

The main cornerstone of the philosophy underlying the μTOSS software is the idea of creating an open system. Here, "open" is to be understood in two directions: open-source implementation, and ease of extensibility. Concretely, the software is realized as two R packages, `mutoss` and `mutossGUI`, which can freely be downloaded from R-Forge, see `http://mutoss.r-forge.r-project.org/`. Every included method comes with a precise description of its usage, its assumptions and appropriate references. This is meant to be of help both to programmers who want to extend μTOSS and to end users who are typically not experts in the vast diversity of existing MHT procedures. In particular, the GUI features an automatic filtering of adequate procedures based on the user's input, and presents the documentation of each procedure in an information window (see Section 4 for details). The components of the μTOSS system provide

(i) multiple tests controlling the family-wise error rate (single-step and stepwise rejective methods, resampling-based procedures),

(ii) multiple tests controlling the false discovery rate (classical and data-adaptive frequentistic methods as well as Bayesian approaches and resampling-based techniques),

(iii) estimation techniques for the number (or proportion) of true null hypotheses,

(iv) multiplicity-adjusted (parametric and non-parametric) simultaneous confidence intervals.

Emphasis has been put on having a broad coverage of available methods, so that μTOSS includes both classical procedures and recently developed MHT procedures. In particular, the package provides μTOSS-compatible functions which serve as interfaces to the standard packages `multcomp` and `multtest`. In addition, very recent algorithms for constructing nonparametric confidence intervals as published in Konietschke (2009) and Gao et al. (2008) and implicitly adaptive FDR-controlling step-up-down tests taken from Finner et al. (2009), Finner et al. (2010) and Blanchard and Roquain (2009) are made available in μTOSS for the first time.

2.2. High-level description

Standard μTOSS specification for functions

One of the first goals in developing the μTOSS package is simply to provide a unified and standardized set of input and output parameters, shared by all procedures that can be called by the user. Since those can cover a large variety of very different purposes, any single procedure actually only uses a subset of all possible input parameters, and returns a subset of possible output parameters. The important point is that procedures which are comparable in their purpose (for example: procedures which return a list of rejected hypotheses based on a list of p-values and a prescribed FDR level) will have the same form of input and output.

A function following this specification, in the sequel called a μTOSS-compatible function, is simply an `R` function taking input parameters under their standard names (and possibly other procedure-specific tuning parameters), and returning as output a list of results also following the standard name spefication. Such functions can then be accessed:

1. as simple command-line `R` functions
2. by applying them on objects of class `Mutoss` through the wrapper function `mutoss.apply` (see below)
3. through the GUI (see Section 4).

The object class `Mutoss`

The class `Mutoss` provides a structure where all the possible standard input/output parameters of the μTOSS specification are present as slots:

```
> slotNames("Mutoss")
 [1] "data"           "model"          "description"    "statistic"
 [5] "hypotheses"     "hypNames"       "criticalValues" "pValues"
 [9] "adjPValues"     "errorControl"   "rejected"       "qValues"
[13] "locFDR"         "pi0"            "confIntervals"  "commandHistory"
```

The wrapper function `mutoss.apply` (which takes as argument an object of class `Mutoss` and a μTOSS-compatible function) reads the object's relevant slots to be fed as the function input, and fills or replaces the object's slots whose names are present in the function output list. This construction allows to follow a consistent and unified processing flow by applying in succession several different processing functions to an object. In this perspective, it also keeps track of useful side information in the process: command history is recorded in a special slot, and for each individual slot, the name of the last function that modified its value is stored.

Statistical models

Multiple testing procedures can be used for a wide variety of different statistical models. This variety prevents from defining a set of unique parameters which could cover the model specification in all cases. Consequently, we introduced a generic input slot, "model", which should contain a structure with all relevant information pertaining to the model definition.

In turn, this structure must follow a certain standard model-dependent specification; at this point we have defined such parameter specifications for several standard models such as one- and two-sample t-tests, F-tests, tests for linear contrasts, etc. Given the model specification, it is possible to compute a list of marginal p-values from the raw data. While some generic MHT procedures can be applied to any list of marginal p-values, other procedures can only be applied for a specific model (but are then often more efficient than generic procedures).

Extensibility
Given the architecture described in the previous paragraph, it is straightforward to add new processing functions to the package: basically it is only necessary to provide a function whose input and output are μTOSS compatible. In addition, in order for this function to be correctly accessible in the GUI, it is necessary to provide an information function having the name `mutoss.<functionname>`, and whose role is to provide metadata about the original function, namely: input and output parameters used, and a HTML-formatted description text (see Section 4.1 for more details).

Example
We demonstrate very briefly the use of the Benjamini-Hochberg adaptive procedure (cf. Benjamini and Hochberg (2000)) using both the direct call to the function or the `mutoss.apply` strategy.

```
> pval = c(runif(95),0.005*runif(5))    # some toy p-values
> BHoutput = adaptiveBH(pValues = pval, alpha = 0.05)

                Benjamini-Hochberg (2000) adaptive step-up procedure

Number of hyp.:  100
Number of rej.:  2
  rejected      pValues adjPValues
1       96 0.0001267805 0.01229771
2       97 0.0006986036 0.03388228
> newObject <- mutoss.apply(new(Class="Mutoss", pValues=pval), f=adaptiveBH,
+                           alpha=0.05, silent=T)
+                           #same thing as above, through mutoss.apply
> which(newObject@rejected)
[1] 96 97
```

3. Simulation platform

The standardization of input/output parameters is of primary importance in order to set up comparison benchmarks between different procedures in an easy manner. This is of use on the one hand for users wanting to explore the output of different methods on a given dataset, and on the other hand for developers of new methodology who want to compare the performance of their method against reference procedures on simulated data.

Given the importance of the latter use case, functions for facilitating a large simulation setup are included as part of μTOSS. The simulation platform consists of just two functions, `simulation()` and `gatherStatistics()`, which are essentially automating loop work for the user. They can be generally thought of as performing Map/Reduce type operations. At this point, the corresponding loops are performed sequentially, but future development should include parallelization support if available.

The `simulation()` function takes three input arguments: the number of replications; the data generating function and its parameters; a list of processing functions and their parameters. The data generating function must be provided by the user and follow a standard specification of its output which is comparable to that of μTOSS compatible functions. The parameters used in arguments 2 and 3 can in general themselves be lists of values to be looped over.

What the `simulation()` function does is to call the data generating function for each possible parameter configuration coming from the parameter list of argument 2. The resulting data is then fed to each of the procedures listed in argument 3, and for each possible parameter configuration coming from their respective parameter lists. This is also repeated over the number of replications.

The output of `simulation()` is a list of objects of the class `MutossSim` (an extension of the class `Mutoss` discussed previously), each object storing the speficic simulation parameter configuration (for data generation as well as processing) used to generate it, along with the result of the procedure.

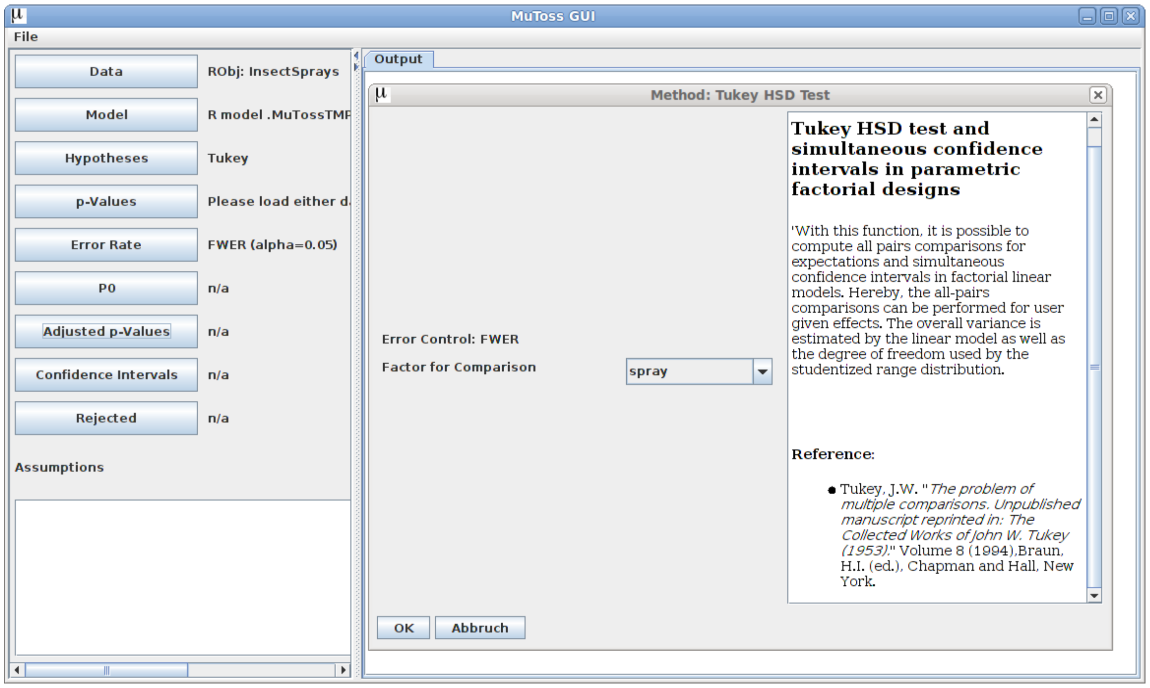
The `gatherStatistics()` function also takes three input arguments: a list of objects created by `simulation()`, a list of postprocessing functions to be applied to each single object, and a list of postprocessing functions to be applied to each group of objects.

The second argument is typically made of numeric output functions; each of these functions is applied to each object appearing in the first argument's list (for example: calculation of the number of false positives in each simulation run). A "group of objects" is the set of objects sharing an identical simulation parameter configuration. Each function appearing in argument 3 (for example: mean or variance) is applied separately for each object group, and to each output column generated by the single object functions. If the third argument is non-void, the output data frame has only a single row for each unique simulation parameter configuration (hence its characterization as a "reduce" step).

The output of `gatherStatistics()` is a data frame, having one column per simulation parameter for recalling their value, and one column for each of the possible combinations of (single object postprocessing, group postprocessing). This data frame can be used in a variety of ways to visualize the outputs using standard libraries in `R`.

4. Graphical user interface

The μTOSS graphical user interface is essentially a visual representation of the underlying concept described in Section 2.2. Its leftmost part (cf. Figure 1) consists of a series of buttons the functionality of which is to modify the associated slots of the active object of class "Mutoss". A filtering mechanism (cf. Section 4.1 for details) guides the user to appropriate functions. After the user has specified the input data set, a statistical model, a family of hypotheses to be tested and the type and level of error control, (s)he can



Data RObj: InsectSprays
Model R model .MuTossTMP
Hypotheses Tukey
p-Values Please load either d
Error Rate FWER (alpha=0.05)
P0 n/a
Adjusted p-Values n/a
Confidence Intervals n/a
Rejected n/a

Error Control: FWER
Factor for Comparison spray

Tukey HSD test and simultaneous confidence intervals in parametric factorial designs

'With this function, it is possible to compute all pairs comparisons for expectations and simultaneous confidence intervals in factorial linear models. Hereby, the all-pairs comparisons can be performed for user given effects. The overall variance is estimated by the linear model as well as the degree of freedom used by the studentized range distribution.

Reference:

- Tukey, J.W. "*The problem of multiple comparisons. Unpublished manuscript reprinted in: The Collected Works of John W. Tukey (1953).*" Volume 8 (1994),Braun, H.I. (ed.), Chapman and Hall, New York.

Figure 1: Screenshot of the μTOSS GUI

select an appropriate method by clicking on either one of the three buttons “adjusted p-values”, “confidence intervals” or “rejected”. According to the information entered before, only procedures that match these specifications can be selected.

When a procedure is selected, a subwindow provides additional information on the selected method and possible further arguments or parameters can be entered or selected (an example is shown in Figure 1). Once all computations are done, the results are displayed in the right part of the GUI. Results are either presented in textual form (listings) or in graphical form (charts). They can be saved as `R` objects or printed into a PDF document.

4.1. Filtering mechanism

The filtering mechanism in the μTOSS GUI is based on the wrapper function class `MutossMethod`. The contents of the slots of a ”MutossMethod” object provide the GUI with all necessary information for realizing the filtering. The following slots are available.

```
> slotNames("MutossMethod")
[1] "label"          "errorControl" "callFunction"   "output"   "info"
[6] "assumptions"  "parameters"   "misc"
```

To illustrate the process, assume that a function `foo` exists the output of which can be used to fill certain slots of a ”Mutoss” object. Then, all that is to be done in order to make `foo` accessible via the μTOSS GUI is to write a wrapper function named `mutoss.foo` returning an object of type `MutossMethod`. The ”label” slot should contain a string with the name of the procedure `foo` is implementing, "callFunction" should equal `foo` and the other self-explaining slots should contain the meta information about the procedure realized by `foo`. Every time the μTOSS GUI is invoked, it screens the `R` workspace for functions with the prefix `mutoss.` and automatically enters them.

5. Outlook

It is envisioned to include modules for graph-based hierarchical testing and for planning and evaluating group-sequential and adaptive (clinical) trials in subsequent releases of μTOSS. Ongoing maintenance is assured by making use of the bug-tracking functionality provided by R-Forge. Subsequent extensions of μTOSS are planned depending on the feedback from the scientific community. The general aim thereby consists in establishing μTOSS as a state of the art in statistical computing for MHT.

Acknowledgments

We like to acknowledge PASCAL2 and especially Nicola Cancedda for support and the Machine Learning / Intelligent Data Analysis group at Berlin Institute of Technology for hosting the core developing phase of μTOSS.

References

Y. Benjamini. Simultaneous and selective inference: current successes and future challenges. Keynote lecture, 6th International Conference on Multiple Comparison Procedures, Tokyo, 2009.

Y. Benjamini and Y. Hochberg. On the adaptive control of the false discovery rate in multiple testing with independent statistics. *J. Edu. and Behav. Stat.*, 25:60–83, 2000.

G. Blanchard and E. Roquain. Adaptive FDR control under independence and dependence. *Journal of Machine Learning Research*, 10:2837–2871, 2009.

H. Finner, T. Dickhaus, and M. Roters. On the false discovery rate and an asymptotically optimal rejection curve. *The Annals of Statistics*, 37:596–618, 2009.

H. Finner, V. Gontscharuk, and T. Dickhaus. FDR controlling step-up-down tests related to the asymptotically optimal rejection curve. *Under review*, 2010.

X. Gao, M. Alvo, J. Chen, and G. Li. Nonparametric multiple comparison procedures for unbalanced one-way factorial designs. *Journal of Statistical Planning and Inference*, 138: 2574–2591, 2008.

F. Konietschke. Simultane Konfidenzintervalle für nichtparametrische relative Kontrasteffekte. Ph. D. dissertation, Georg-August University, Göttingen, 2009.

U. S. Food and Drug Administration. Draft guidance on "adaptive design clinical trials for drugs and biologics". `http://www.fda.gov/Drugs/GuidanceComplianceRegulatoryInformation/Guidances/ucm121568.htm`, 2010.