Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond

Achim Zeileis
Universität Innsbruck

Nikolaus Umlauf
Universität Innsbruck

Friedrich Leisch
Universität für Bodenkultur Wien

Abstract

The capabilities of the package **exams** for automatic generation of (statistical) exams in R are extended by adding support for learning management systems: As in earlier versions of the package exam generation is still based on separate `Sweave` files for each exercise – but rather than just producing different types of PDF output files, the package can now render the *same* exercises into a wide variety of output formats. These include HTML (with various options for displaying mathematical content) and XML specifications for online exams in learning management systems such as **Moodle** or **OLAT**. This flexibility is accomplished by a new modular and extensible design of the package that allows for reading all weaved exercises into R and managing associated supplementary files (such as graphics or data files). The manuscript discusses the readily available user interfaces, the design of the underlying infrastructure, and how new functionality can be built on top of the existing tools.

Keywords: exams, e-learning, multiple choice, arithmetic problems, `Sweave`, R, LaTeX, HTML, XML, IMS QTI, **Moodle**, **OLAT**.

1. Introduction

The design for version 1 of the **exams** package was conceived seven years ago (in 2006) when the original authors (Grün and Zeileis 2009) were involved in a redesign of the introductory statistics lecture at WU Wirtschaftsuniversität Wien. Back then the main goal was to be able to produce exams along with associated self-study materials as PDF (portable document format) files. Thus, the main focus was still on printable materials for classic classroom exams. Although e-learning systems started to become available more easily back at that time, they were still not very widely used and, more importantly, rather few easy-to-use standards for specifying e-learning exams were available (e.g., WU Wien used a partially self-developed e-learning system based on **.LRN**, see Blesius *et al.* 2007).

However, since 2006 the situation has clearly changed: E-learning systems are now abundant with many universities offering one (ore more) e-learning system(s) in which all students are readily registered. Consequently, many lecturers routinely offer online exams (or tests, quizzes, assessments) for large-lecture courses – either as self-study materials or as (part of) the main assessment of the course.

Among the more popular choices of learning management systems are the open-source systems

Moodle – developed by Dougiamas *et al.* (2013) and supported by a large world-wide user community – or **OLAT** (for online learning and training) – originally developed by Universität Zürich (2012) and with the recent fork **OpenOLAT** being developed by frentix GmbH (2013) and a support community. A popular proprietary learning management system is **Blackboard** developed by (**Blackboard** Inc. 2010). Standards for specifying and exchanging e-learning exams/assessements have also emerged (see Agea *et al.* 2009, for an overview). While **Moodle** specifies its own **Moodle** XML format (but can import and export several other formats), **OLAT**/**OpenOLAT** and **Blackboard** employ certain subsets of the international QTI (question & test interoperability) standard, version 1.2, maintained by the IMS Global Learning Consortium, Inc. (2002). The successor formats are QTI 2.0 and the current QTI 2.1 which is for example employed in the **ONYX** testsuite (BPS Bildungsportal Sachsen GmbH 2013) that also offers interfaces to **OLAT** and **Blackboard**.

Therefore, although the PDF exams produced by version 1 of the **exams** package as introduced by Grün and Zeileis (2009) are still useful for many types of courses, it would also be highly desirable to have support for generating e-learning exams *from the same pool of exercises*. In fact, this became an apparent need when the authors of the present manuscript took over new large-lecture statistics and mathematics courses at their respective institutions (Universität Innsbruck and Universität für Bodenkultur Wien, BOKU, respectively). For example, the new "Mathematics 101" lecture at Universität Innsbruck is currently attended by about 1,600 students (mostly first-year business and economics students) and accompanied by bi-weekly online exams conducted in the university's **OLAT** learning management system. This was a strong incentive to start developing version 2 of the **exams** package that is presented here and offers an extensible toolbox for generating e-learning exams, including easy-to-use functions for **Moodle** quizzes and **OLAT** assessments[1].

The new version of the **exams** package for the R system for statistical computing (R Core Team 2013) is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=exams`. Like prior versions it employs ideas and technologies from literate programming and reproducible research (see e.g., Knuth 1992; de Leeuw 2001; Leisch and Rossini 2003; Kuhn 2013) by using `Sweave()` (Leisch 2002) to combine data-generating processes in R with corresponding questions/solutions in LaTeX (Knuth 1984; Lamport 1994). But in addition to producing exams in PDF format, the new version of **exams** includes extensible tools for generating other output formats *without* having to modify the pool of exercises. Thus, the design principles of the **exams** package are only somewhat extended compared to version 1:

- Each exercise template (also called "exercise" for short) is a single `Sweave` file (`.Rnw`) interweaving R code for data generation and LaTeX code for describing question and solution (possibly containing mathematical notation in LaTeX).
- Exams can be generated by randomly drawing different versions of exercises from a pool of such `Sweave` exercise templates. The resulting exams can be rendered into various formats including PDF, HTML, **Moodle** XML, or QTI 1.2 (for **OLAT**/**OpenOLAT**).
- Maintenance is simple as exercises are separate standalone files. Thus, large teams can work jointly on the pool of exercises in a multi-author and cross-platform setting because each team member can independently develop and edit a single exercise.

[1]Currently, **OLAT** and **OpenOLAT** do not differ with respect to their specification of exams. Hence, essentially all discussion of **OLAT** in this manuscript applies to both **OLAT** and **OpenOLAT**.

The remainder of this paper consists of two major parts: First, we illustrate in Section 2 how to use both the old and new exam-generating functions that are readily available in the package. This serves as a first introduction and is sufficient for getting a good overview of the available features and how to get started. Second, Section 3 provides details about the design underyling the toolbox for the new infrastructure. This section – as well as the subsequent Section 4 showing how to extend the toolbox – may be skipped upon first reading but it contains many important details that are likely to be required when actually starting to create course materials with the package. Finally, a discussion in Section 5 concludes the paper.

2. Using the exams package

In this section we provide an overview of the most important user interfaces provided by the **exams** package. This serves as a first introduction, assuming only (basic) knowledge of `Sweave` (Leisch 2012a,b). First, the format of the exercise `Sweave` files is reviewed along with the old (version 1) `exams()` function. Subsequently, the new (version 2) functions of type `exams2`*xyz*`()` are introduced: `exams2pdf()` and `exams2html()` produce one PDF or HTML file for each exam, respectively. In case of just a single exam, this is shown interactively in a viewer/browser. `exams2moodle()` and `exams2qti12()` generate **Moodle** and QTI 1.2 exams, i.e., just a single XML or ZIP file, respectively, which can be easily uploaded into **Moodle** and **OLAT**.

2.1. Version 1: PDF `exams()` from `Sweave` exercises

Exercise templates (or just "exercises" for short) are essentially separate standard `Sweave` files (Leisch 2012a,b). They are composed of the following elements:

- R code chunks (as usual within `<<>>=` and `@`) for random data generation.
- Question and solution descriptions contained in LaTeX environments of corresponding names. Both can contain R code chunks again or include data via `\Sexpr{}`.
- Metainformation about the exercise type (numeric, multiple choice, ...), its correct solution etc. All metainformation commands are in LaTeX style but are actually commented out and hidden in the final output file.

The underlying ideas are eplained in more detail by Grün and Zeileis (2009) and Section 3 provides more technical details. Here, we focus on illustrating how different output formats can be generated from such exercises.

In Figure 1, the `Sweave` file for a simple exercise asking students to compute a one-sample t test statistic is shown for illustration (as already used by Grün and Zeileis 2009). The R chunk for the data-generating process (DGP), the `question` and `solution` environments, and the metainformation can be easily distinguished. The LaTeX file resulting from an `Sweave()` call is shown in Figure 2, and Figure 3 shows the final compiled PDF output generated by

```
R> library("exams")
R> set.seed(1090)
R> exams("tstat.Rnw")
```

```
<<echo=FALSE, results=hide>>=
## DATA GENERATION
n <- sample(120:250, 1)
mu <- sample(c(125, 200, 250, 500, 1000), 1)
y <- rnorm(n, mean = mu * runif(1, min = 0.9, max = 1.1),
           sd = mu * runif(1, min = 0.02, max = 0.06))

## QUESTION/ANSWER GENERATION
Mean <- round(mean(y), digits = 1)
Var <- round(var(y), digits = 2)
tstat <- round((Mean - mu)/sqrt(Var/n), digits = 3)
@

\begin{question}
  A machine fills milk into $\Sexpr{mu}$ml packages. It is suspected that the
  machine is not working correctly and that the amount of milk filled differs
  from the setpoint $\mu_0 = \Sexpr{mu}$. A sample of $\Sexpr{n}$ packages
  filled by the machine are collected. The sample mean $\bar{y}$ is equal to
  $\Sexpr{Mean}$ and the sample variance $s^2_{n-1}$ is equal to
  $\Sexpr{Var}$.

  Test the hypothesis that the amount filled corresponds on average to the
  setpoint. What is the absolute value of the $t$~test statistic?
\end{question}

\begin{solution}
  The $t$~test statistic is calculated by:
  \begin{eqnarray*}
    t & = & \frac{\bar y - \mu_0}{\sqrt{\frac{s^2_{n-1}}{n}}}
    = \frac{\Sexpr{Mean} - \Sexpr{mu}}{\sqrt{\frac{\Sexpr{Var}}{\Sexpr{n}}}}
    = \Sexpr{tstat}.
  \end{eqnarray*}
  The absolute value of the $t$~test statistic is thus equal to
  $\Sexpr{format(abs(tstat), nsmall = 3)}$.
\end{solution}

%% META-INFORMATION
%% \extype{num}
%% \exsolution{\Sexpr{format(abs(tstat), nsmall = 3)}}
%% \exname{t statistic}
%% \extol{0.01}
```

Figure 1: A simple `Sweave` exercise: `tstat.Rnw`.

```
\begin{question}
  A machine fills milk into $500$ml packages. It is suspected that the
  machine is not working correctly and that the amount of milk filled differs
  from the setpoint $\mu_0 = 500$. A sample of $226$ packages
  filled by the machine are collected. The sample mean $\bar{y}$ is equal to
  $517.2$ and the sample variance $s^2_{n-1}$ is equal to
  $262.56$.

  Test the hypothesis that the amount filled corresponds on average to the
  setpoint. What is the absolute value of the $t$~test statistic?
\end{question}

\begin{solution}
  The $t$~test statistic is calculated by:
  \begin{eqnarray*}
    t & = & \frac{\bar y - \mu_0}{\sqrt{\frac{s^2_{n-1}}{n}}}
    = \frac{517.2 - 500}{\sqrt{\frac{262.56}{226}}}
    = 15.958.
  \end{eqnarray*}
  The absolute value of the $t$~test statistic is thus equal to
  $15.958$.
\end{solution}

%% META-INFORMATION
%% \extype{num}
%% \exsolution{15.958}
%% \exname{t statistic}
%% \extol{0.01}
```

Figure 2: LaTeX output of `Sweave("tstat.Rnw")`.

Here, the `exams()` function looks for the exercise template `tstat.Rnw` first in the local working directory and then within the installed **exams** package where this file is provided. Then it copies the exercise `.Rnw` to a temporary directory, calls `Sweave()` to generate the `.tex`, and includes this in the default LaTeX template for exams before producing the `.pdf`. As, by default, just a single `.pdf` exam is produced and no output directory is specified, a PDF viewer pops up and displays the resulting exam (as in Figure 3).

While applying `exams()` to just a single exercise is very useful while writing/programming an exercise, a full exam will typically encompass several different exercises. Also, it may require suppressing the solutions, including a title page with a questionnaire form, etc. The former can be achieved by supplying a (list of) vector(s) of exercises while the latter can be accomodated by using different templates:

1. **Problem**
   A machine fills milk into 500ml packages. It is suspected that the machine is not working correctly and that the amount of milk filled differs from the setpoint $\mu_0 = 500$. A sample of 226 packages filled by the machine are collected. The sample mean $\bar{y}$ is equal to 517.2 and the sample variance s^2_{n-1} is equal to 262.56.

   Test the hypothesis that the amount filled corresponds on average to the setpoint. What is the absolute value of the t test statistic?

   Solution
   The t test statistic is calculated by:

$$t = \frac{\bar{y} - \mu_0}{\sqrt{\frac{s^2_{n-1}}{n}}} = \frac{517.2 - 500}{\sqrt{\frac{262.56}{226}}} = 15.958.$$

   The absolute value of the t test statistic is thus equal to 15.958.

Figure 3: Display of a `tstat` exercise as PDF via `exams()` (or `exams2pdf()`).

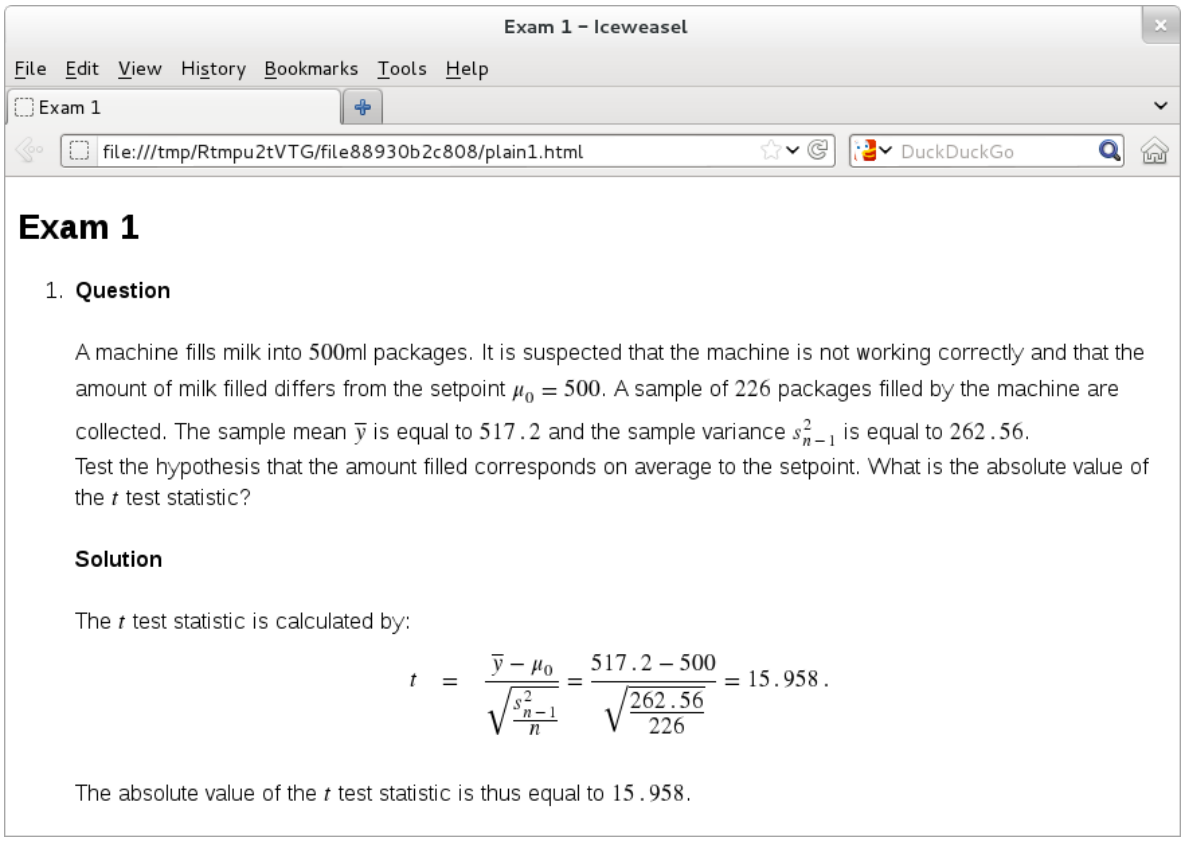

Figure 4: Display of a `tstat` exercise as HTML via `exams2html()`. MathML is employed for mathematic equations, as rendered by a **Firefox** browser.

Argument	Description
`file`	A (list of) character vector(s) specifying the (base) names of the `Sweave` exercise files.
`n`	The number of exams to be generated from the list of exercises. Default: 1.
`nsamp`	The number of exercise files sampled from each list element of `file`. Default: One for each list element.
`dir`	Path to output directory. Default: Single PDF or HTML files are shown directly in a viewer/browser (i.e., `exams`/`exams2pdf`/`exams2html` with `n = 1`). In all other cases the current working directory is used by default.
`edir`	Path to the directory in which the exercises in `file` are stored. Default: Working directory (or within the **exams** installation).
`tdir`	Path to a temporary directory in which `Sweave()` is carried out. Default: New `tempdir()`.
`sdir`	Path to the directory in which supplementary files (e.g., graphics or data files) are stored (except for `exams()`). Default: New `tempdir()`.
`name`	Name prefix for the resulting exam files.
`template`	Character specifying the (base) names of a LaTeX, HTML, or XML file template for the exam (except for `exams2moodle()`). Default: A function-specific template provided within the **exams** installation.
`encoding`	Character specifying the encoding to be used (in version 2 interfaces only).

Table 1: Common arguments of the main user interfaces for generating exams: `exams()`, `exams2pdf()`, `exams2html()`, `exams2moodle()`, `exams2qti12()`. The first group of arguments pertains to the specification of the exam(s), the second group to the handling of input/temporary/output directories, and the last group to name and setup for the resulting files. For further function-specific arguments and more details/examples, see the corresponding manual pages.

```
R> myexam <- list(
+    "boxplots",
+    c("confint", "ttest", "tstat"),
+    c("anova", "regression"),
+    "scatterplot",
+    "relfreq")
R> odir <- tempfile()
R> set.seed(1090)
R> x1 <- exams(myexam, n = 3, dir = odir, template = c("exam", "solution"))
```

The `myexam` list contains five exercises: the first one is always `boxplots.Rnw` while the second exercise is randomly drawn from `confint.Rnw`, `ttest.Rnw`, `tstat.Rnw`, and so on for the remaining exercises. Then, `exams()` is used to draw `n = 5` random exams and produce one exam and one solution PDF for each. The `template` argument takes names of LaTeX files which provide the LaTeX headers and footers. These templates can be used to create a title page with a questionnaire form (for student name, id, signature, etc.), show or suppress solutions, and set further formatting details. All involved `.Rnw` files (with exercises) and `.tex` templates employed in the example above are provided in the **exams** source package and its installed versions. The resulting output files are stored along with the extracted metainformation in

the output directory:

```
R> dir(odir)

[1] "exam1.pdf"     "exam2.pdf"     "exam3.pdf"     "metainfo.rda"
[5] "solution1.pdf" "solution2.pdf" "solution3.pdf"
```

More details on basic usage and more advanced customization of this function are provided by Grün and Zeileis (2009), also provided in an updated version as `vignette("exams", package = "exams")`. The latter also discusses some small changes and enhancements made for `exams()` (and `exams2pdf()`, respectively). An overview of the most important arguments that are also shared by the version 2 interfaces is given in Table 1.

2.2. Version 2: Producing PDF, HTML, or XML for Moodle or OLAT

The new infrastructure added to the **exams** package on the road to version 2 is providing more flexibility and enables a much broader variety of output formats while keeping the specification of the exercise templates fully backward compatible and only slightly extended. While the design of the underlying workhorse functions is rather different (see Section 3), the new user interfaces are very similar to the old one, sharing most of its arguments (see Table 1). Hence, for users of the previous version of the package, it is easy and straightforward to adapt to the new facilities.

Producing PDF documents: `exams2pdf()`

The function `exams2pdf()` is mainly a proof-of-concept reimplementation of `exams()` using the new extensible infrastructure of the **exams** package. For the user virtually nothing changes:

```
R> set.seed(1090)
R> exams2pdf("tstat.Rnw")
```

pops up the same PDF as shown in Figure 3. We refrain from further discussion of customization of the PDF output because this is discussed in Grün and Zeileis (2009) with details about LaTeX master templates, additional auxiliary files, showing/hiding solutions etc. Here we only point out the main difference between the old `exams()` function and the new `exams2pdf()`: The latter not only returns the metainformation from the exercise but additionally also the LaTeX code for the question and solution environments as well as paths to supplementary materials (such as graphics or data files). Section 3 explains the structure of the return values in more detail and illustrates how this can be used.[2]

Producing HTML documents: `exams2html()`

As a first step towards including exams generated from `Sweave` files into e-learning exams, it is typically necessary to be able to generate an HTML version of the exams. Hence, the function `exams2html()` is designed analogously to `exams()`/`exams2pdf()` but produces HTML files. In case of just a single generated exam, this is displayed in a browser using base

[2] To obtain the same type of return value as from the `exams()` function, `exams_metainfo(exams2`*xyz*`(...))` can be used.

R's `browseURL()` function[3]. Again, this is particularly useful while writing/programming a new exercise template. For example,

```
R> set.seed(1090)
R> exams2html("tstat.Rnw")
```

generates the HTML file shown in Figure 4 which corresponds directly to the PDF file from Figure 3. Note that for properly viewing the formulas in this HTML file, a browser with MathML support is required. This is discussed in more detail in Section 3.4. Here, **Iceweasel** is used – Debian Linux's rebranding of the **Firefox** browser which has native MathML support.

To transform the LaTeX questions/solutions to something that a web browser can render, three options are available: translation of the LaTeX to (1) plain HTML, (2) HTML plus MathML for mathematical formulas (default), or conversion of the corresponding PDF to (3) HTML with one embedded raster images for the whole question and solution, respectively. The former two options are considerably faster and more elegant – they just require the R package **tth** (Hutchinson, Leisch, and Zeileis 2013) that makes the 'TeX-to-HTML' converter **TtH** (Hutchinson 2012) easily available in R. Also, by default, the **base64enc** package (Urbanek 2012) is employed for embedding graphics in Base64 encoding. More details on this approach are provided in Section 3.4.

The HTML files produced with approaches (1) and (2) can also easily contain hyperlinks to supplementary files. For example, if the R code in the `Sweave` file generates a file `mydata.rda`, say, then simply including `\url{mydata.rda}` in the question/solution will result in a suitable hyperlink. The supplementary data files for each random replication of the exercise is managed fully automatically and a copy of the data is created in an (exam-specific) sub-directory of the output directory. Run `exams2html("boxhist.Rnw")` for such an example.

Just like `exams()`/`exams2pdf()`, `exams2html()` can also generate multiple replications of randomly drawn exams via `exams2html(myexam, n = 3, dir = odir)`. Also multiple versions of the same replications can be generated by providing several templates, e.g., for showing/suppressing solutions.

Producing ***Moodle*** *XML:* `exams2moodle()`

To incorporate exams generated from `Sweave` exercises into learning management systems, such as **Moodle**, two building blocks are typically required: (1) questions/solutions are available in plain text or HTML format, and (2) questions/solutions can be embedded along with the metainformation about the possible and correct solutions into some exam description format. (1) can be accomplished as outlined in the previous subsection for `exams2html()` and for **Moodle** (2) requires embedding everything into **Moodle** XML format. Both steps can be easily carried out using the `exams2moodle()` function:

```
R> set.seed(1090)
R> exams2moodle(myexam, n = 3, dir = odir)
```

This draws the same three random exams from the `myexam` list that were already generated in PDF format above. The output file, stored again in `odir`, is a single XML file.

[3]In **RStudio** (**RStudio** Team 2013), versions prior to 0.97.133, the `"browser"` option is set to a function that cannot browse local HTML files on some platforms. Recent versions of **RStudio** have resolved this problem and `?exams2html` also provides workarounds for older **RStudio** versions.

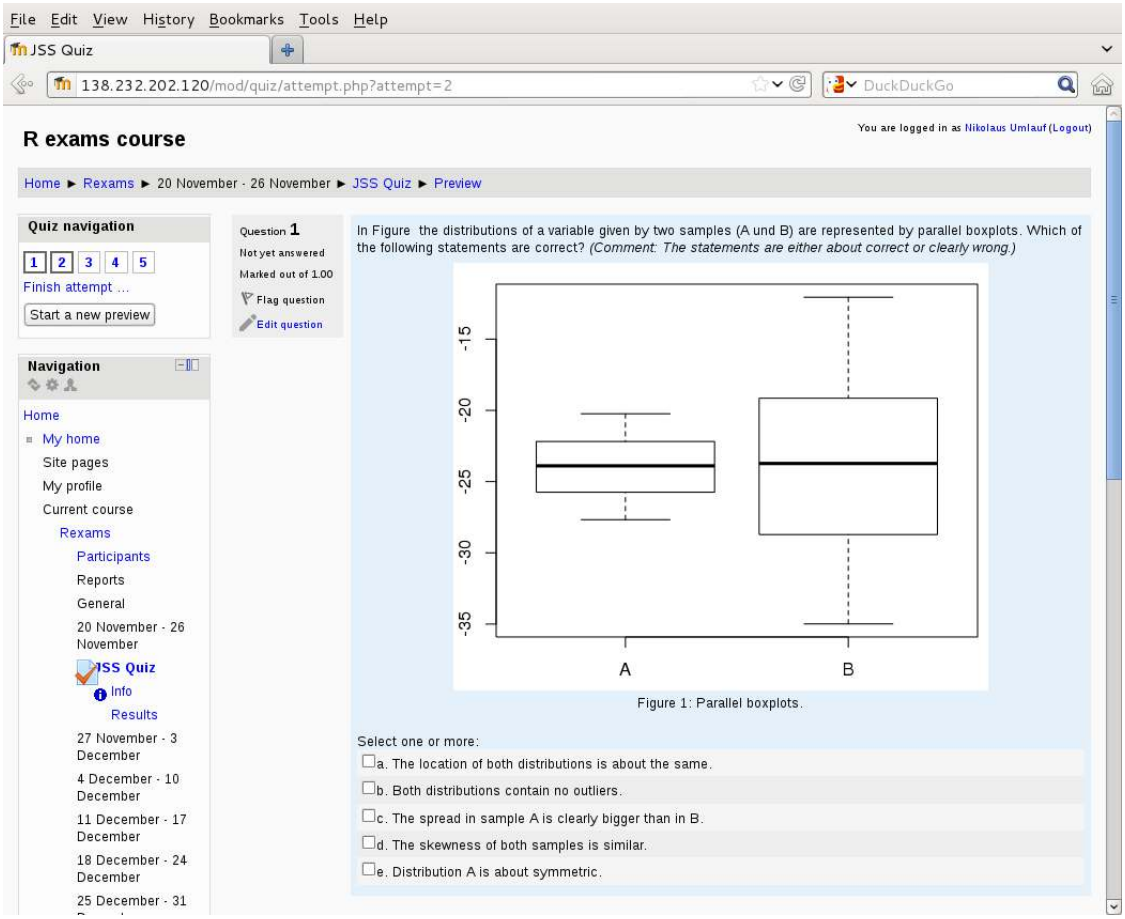

Figure 5: Display of exercise 1 (`boxplots`) from `myexam` in **Moodle** (as rendered by a **Firefox** browser).

```
R> dir(odir)
```

```
[1] "exam1.pdf"       "exam2.pdf"       "exam3.pdf"       "metainfo.rda"
[5] "moodlequiz.xml" "solution1.pdf"  "solution2.pdf"  "solution3.pdf"
```

This XML file `moodlequiz.xml` can be easily imported into a **Moodle** quiz and then further customized: First, the XML file is imported into the question bank in **Moodle**. Then, all replications of each exercise can be added as "random" questions into a quiz (and potentially further customized). Figure 5 shows the first random draw of the `boxplots` exercise in the resulting **Moodle** quiz (again rendered by a **Firefox** browser). More details on how **exams**-generated questions can be integrated in **Moodle** are provided in Section 5.

The corresponding solutions are displayed upon completion of the exam in **Moodle**. As before, selected supplementary files are automatically managed and can easily be included using `\url{}` in the underlying LaTeX code. To be able to include all these supplements in a single XML file, Base64 encoding is employed for all supplements. See the manual page for the list of all supported supplement file formats.

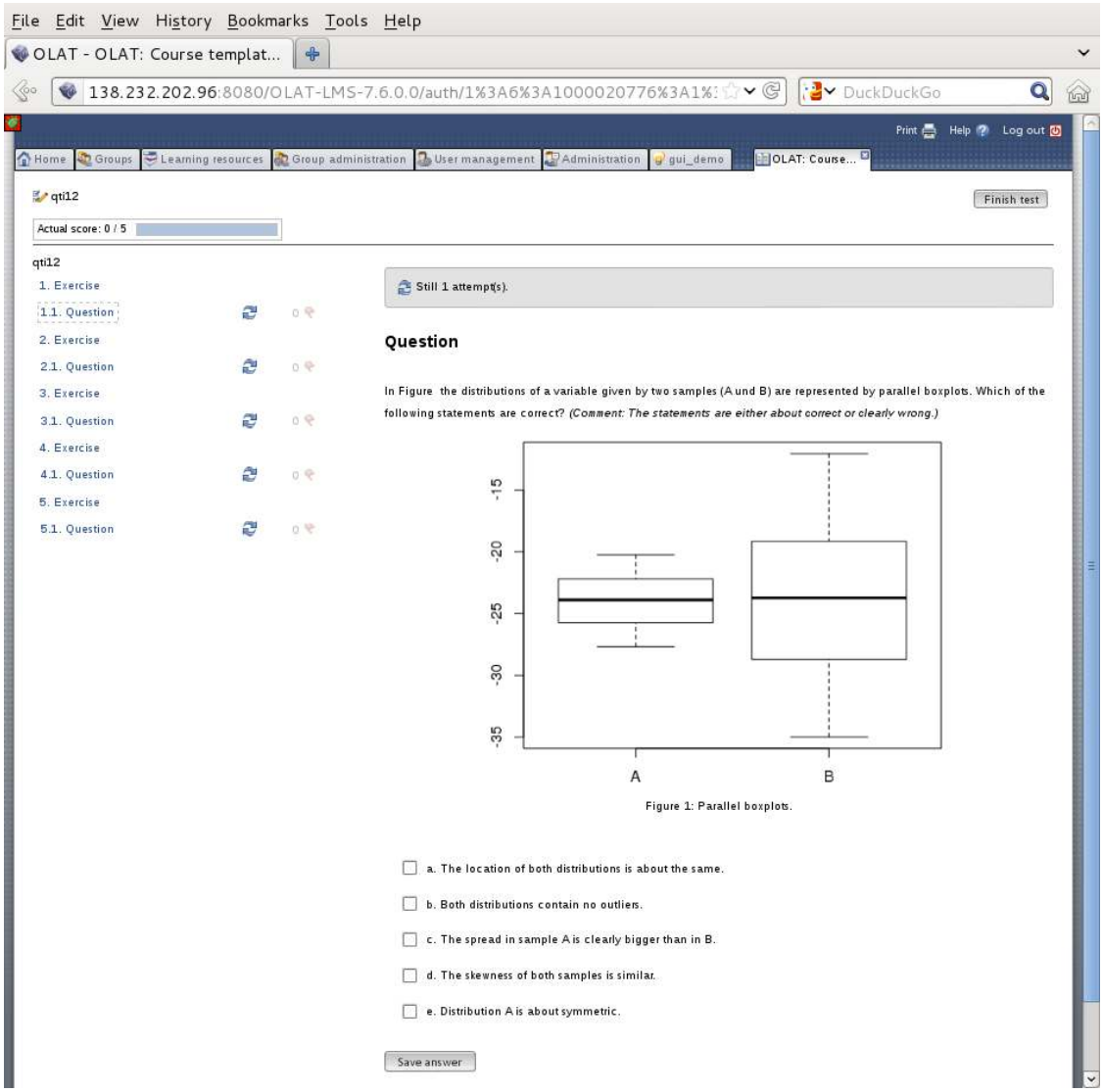

Figure 6: Display of exercise 1 (`boxplots`) from `myexam` in **OLAT** (as rendered by a **Firefox** browser).

*Producing QTI 1.2 XML (for **OLAT**):* `exams2qti12()`

The generation of QTI 1.2 assessments (for **OLAT**) proceeds essentially in the same way as for the **Moodle** quizzes, by default using `ttm` for transformation of the text to HTML[4]. The same three random draws of exams from `myexam` can be prepared in QTI 1.2 format via:

```
R> set.seed(1090)
R> exams2qti12(myexam, n = 3, dir = odir)
```

[4]It may be of interest to **OLAT** users that we experienced problems with the display of MathML matrices in **OLAT**. The columns were not separated by spaces and we have not been able to adapt our **OLAT** installation to avoide this problem. Hence, if we want to display matrices in **OLAT**, we generate them with extra empty columns. The `cholesky` exercise template has code that can automatically do this, if enabled.

This produces a single ZIP file `qti.zip`, again written to `odir`.

```
R> dir(odir)

[1] "exam1.pdf"      "exam2.pdf"      "exam3.pdf"      "metainfo.rda"
[5] "moodlequiz.xml" "qti12.zip"      "solution1.pdf"  "solution2.pdf"
[9] "solution3.pdf"
```

The ZIP file can again be easily imported into an **OLAT** test configuration where further customization can be performed[5]. The first `boxplots` exercise from the exam generated above is shown in **OLAT** in Figure 6 (again as rendered by a **Firefox** browser). The corresponding solutions are displayed in **OLAT** immediately after *incorrectly* completing an individual exercise. The display of solutions can also be suppressed completely by setting `solutionswitch = FALSE` in `exams2qti12()`.[6]

The main difference of the generated ZIP file for QTI 1.2, compared to the **Moodle** XML output, is that in addition to the `qti.xml` file further supplementary files can be included. Hence, supplements in all potential formats can be easily included and uploaded in one go into **OLAT**. Therefore, by default, Base64 is employed only for graphics but not for other files (such as data sets etc.) and can optionally also be disabled if desired.

The QTI 1.2 standard allows for rather fine control of the properties of the exercises (also known as items in QTI 1.2) and the exams (also known as assessments). Hence, `exams2qti12()` provides a variety of options for controlling the appearance of exam/exercises, see the manual page `?exams2qti12` for details. Also, the underlying XML template can be adapted.

2.3. Creating the first exam

When creating the first "real" exam with **exams**, i.e., when starting to prepare course materials with the help of the package, it is our experience that it works best to start (almost) from scratch with some simple examples. The package provides a wide range of examples for typical exercises (see Table 3 in the appendix for an overview) which can serve as a starting point and it is often useful to copy parts of these exercises to create new ones. In particular, we recommend to keep the formatting as simple as possible for two reasons: (1) The resulting exercises are typically more robust and work well with different `exams2`*xyz*`()` interfaces (especially both in LaTeX and in HTML). (2) Some formatting issues require attention to technical details, e.g., as discussed in Section 3.

Thus, we recommend to start with exercises taking inspiration from the available examples and only using basic LaTeX markup for mathematical notation and formatting. To aid this process, **exams** provides the function `exams_skeleton()` (or equivalently `exams.skeleton()`) which creates a directory with a copy of the exercises from Table 3 (in the appendix) and the required templates (e.g., for LaTeX, HTML, or XML) along with a '`demo.R`' script illustrating the use of the various `exams2`*xyz*`()` interfaces.

[5]While customization of the features of the overall assessment was always possible for us, **OLAT** typically did not allow for modification of the individual exercise items. We were not able to track down which part of the QTI 1.2 XML specification causes this.

[6]In our e-learning exams, we typically employ these default settings (i.e., `maxattemps = 1` and `solutionswitch = TRUE`). Alternatively, we give the students an unlimited number of attempts to solve an exercise (`maxattempts = Inf`) but then suppress solutions completely (`solutionswitch = FALSE`) because otherwise the correct solution would be displayed after the first incorrect attempt.

As an illustration, assume that we are interested in using `exams2moodle()` and `exams2pdf()` for generating both **Moodle** and PDF exams (for printout). To test that the LaTeX-to-HTML conversion for **Moodle** actually works for all exercises, we additionally want to inspect the results of `exams2html()`. Hence, the code below calls `exams_skeleton()` specifying these three writers. Here, we employ a temporary directory but users may set the `dir` argument to something like `"C:/myexam/"` or `"~/myexam/"` etc.

```
R> mydir <- file.path(tempdir(), "myexam")
R> exams_skeleton(dir = mydir, absolute = TRUE,
+    writer = c("exams2html", "exams2pdf", "exams2moodle"))
R> dir(mydir)

[1] "demo.R"    "exercises" "templates"
```

The directory then contains the file '`demo.R`' which can be opened in any editor for R scripts. The script illustrates how to create various kinds of output using the `exams2html()`, `exams2pdf()`, and `exams2moodle()` functions based on the exercises and templates in the subdirectories of the same name. Absolute paths are employed in the script to refer to these subdirectories (while the default `absolute = FALSE` would result in relative paths being used).

The function `exams_skeleton()` always copies all exercise files to the directory but the '`demo.R`' script only employs one example for each exercise type, i.e., `num`, `schoice`, `mchoice`, `cloze`, and `string`. To restrict this set of exercises, the `type` argument of `exams_skeleton()` can be used (e.g., `type = c("num", "schoice")`). In any case, it should be easy to modify the '`demo.R`' script, omitting or adding exercises that are readily available in the subdirectory.

Finally, to illustrate how different encodings can be used, `exams_skeleton()` can also be called with an `encoding` argument, e.g., setting `encoding = "UTF-8"`. This modifies the `demo.R` script as well as the HTML and LaTeX templates accordingly. The `encoding`s `"latin1"` (or `"ISO-8859-1"`) and `"latin9"` (or `"ISO-8859-15"`) have also been tested. As usual, employing `Sweave` files in a particular encoding can be very convenient for special characters (such as accents or umlauts) but might also lead to problems if they are used in different locales (e.g., on different operating systems). An alternative route (employed by the authors of the **exams** package) is to employ `Sweave` ASCII files only, using LaTeX commands for special characters.

3. Design

All the new `exams2`*xyz*`()` interfaces for generating exams in different formats (with currently *xyz* $\in$ {`pdf`, `html`, `moodle`, `qti12`}) are built by combining the modular building blocks provided by version 2 of **exams**. All functions have the same goal, i.e., to write exam files in a certain format to the hard disk. The approach is that the `Sweave` exercises are first *weaved* to LaTeX, *read* into R, potentially *transformed* (e.g., to HTML), and then *written* to suitable output file formats on the disk. Different customizable driver functions (or even driver-generating functions) for performing the weave/read/transform/write steps are available in **exams**. Internally, all the `exams2`*xyz*`()` interfaces choose certain drivers and then call the new function `xexams()` (for extensible exams) that handles all temporary files/directories and suitably executes the drivers. In the following subsections, all these building blocks are introduced in detail.

Command	Description
`\extype{}`	Specification of the type of exercise (required): `num` for questions with a numeric answer, `mchoice` for questions with multiple-choice answers, `schoice` for questions with single-choice answers (i.e., multiple-choice with exactly one correct solution), `string` for questions with a (short) text answer, or `cloze` for cloze solutions (i.e., combinations of the above).
`\exname{}`	Short name/description (to be used for printing within R).
`\extitle{}`	Pretty longer title (for **Moodle**).
`\exsection{}`	Section of the exercise (for **Moodle**, with slashes for subsections as in a URL).
`\exversion{}`	Version of the exercise.
`\exsolution{}`	Correct solution (required). It must contain a numeric solution for `num`, a string of zeros/ones for `mchoice`/`schoice`, or a character string of `string`. For `cloze` a combination of these can be specified, e.g., `\exsolution{1.23|001|glm}`.
`\extolerance{}`	Tolerance for `num` solutions or a vector of tolerances (expanded if necessary) for `cloze` solutions. If unspecified the tolerance is 0.
`\exclozetype{}`	List of types for the elements of a `cloze` exercise, e.g., `\exclozetype{num|schoice|string}` for the example above.
`\expoints{}`	Points for correct solution. Default is 1.

Table 2: Overview of metainformation commands in exercises. The commands in the first section allow for a general description, those in the second section for question/answer specification. Only `extype` and `exsolution` are always required (but `exname` is recommended additionally for nice printing in R).

3.1. Extended specification of exercises

As discussed in Section 2 and illustrated in Figure 1, each exercise is simply an `Sweave` file containing R code for data generation, `question`/`solution` environments with LaTeX text, and metainformation about the type of exercise and the correct solution etc. This design was introduced by Grün and Zeileis (2009) but is slightly extended in the new version to provide some more options for the generation of e-learning exams. See Table 2 for an overview for a list of exercise types and corresponding metainformation commands.

Each exercise must specify at least an `\extype{}` and an `\exsolution{}` and should typically also have a short `\exname{}`. There are now five different `extypes`. Two types that have a single question and answer:

- `num` for questions with a numeric answer, e.g., `\exsolution{1.23}`.
- `string` for questions with a (short) text answer, e.g., `\exsolution{glm}`.

Three types have a list of questions (or statements):

- `mchoice` for multiple-choice questions where each element of the question/statement can either be true or false, e.g., `\exsolution{01011}`.

- `schoice` for single-choice questions where exactly one of the questions/statements is true and all others are false, e.g., `\exsolution{01000}`.
- `cloze` for a combination of questions/statements with `num`, `string`, or `mchoice` answers. Thus, each element of the question has either a numeric, short text, or single/multiple-choice answer, e.g., `\exsolution{1.23|001|glm}`. To specify the individual `cloze` types, a `clozetype` has to be given, e.g., `\exclozetype{num|schoice|string}`.

The types `schoice` and `cloze` have been newly introduced. The purpose of the former is mainly to allow for different processing of options (e.g., for assigning points to correct/wrong results) between `mchoice` and `schoice`. The `cloze` type was introduced because both **Moodle** and QTI 1.2 have support for it (albeit in slightly different ways, for details see below). Possible evaluation strategies (with/without partial credits and/or with/without negative points) are discussed below for `exams2moodle()` and `exams2qti12()` and in Appendix B for further functionality within R.

For the three types with lists of questions (`mchoice`, `schoice`, `cloze`), the `question` and `solution` environments should each contain at the end an `answerlist` environment. In the `question` this should list an `\item` for each question/statement and in the `solution` the corresponding answers/explanations can be provided (if any). The `answerlist` environment can either be written as usual "by hand" or by using the `answerlist()` function provided by the **exams** package. For illustration, we set up a multiple-choice question with three statements about Switzerland. First, we generate an `answerlist` with statements for the `question`.

```
R> qu <- c("Zurich is the capital of Switzerland.",
+          "Italian is an official language in Switzerland.",
+          "Switzerland is part of the European Union (EU).")
R> answerlist(qu)

\begin{answerlist}
  \item Zurich is the capital of Switzerland.
  \item Italian is an official language in Switzerland.
  \item Switzerland is part of the European Union (EU).
\end{answerlist}
```

Then the corresponding `answerlist` for the `solution` is set up.

```
R> sol <- c(FALSE, TRUE, FALSE)
R> ex <- c("The capital of Switzerland is Bern.",
+          "The official languages are: German, French, Italian, Romansh.",
+          "Switzerland is part of the Schengen Area but not the EU.")
R> answerlist(ifelse(sol, "True", "False"), ex)

\begin{answerlist}
  \item False. The capital of Switzerland is Bern.
  \item True. The official languages are: German, French, Italian, Romansh.
  \item False. Switzerland is part of the Schengen Area but not the EU.
\end{answerlist}
```

For more examples see the exercise files in the `inst/exercises` directoy of the **exams** source package. There are various multiple-choice questions with and without figures and/or verbatim R output (e.g., `anova`, `boxplots`, `cholesky`, amoung others). The files `tstat` and `tstat2` illustrate how the same type of exercise can be coded as a `num` or `schoice` question, respectively. The `cloze` type is employed in `boxhist` (with more flexible formatting specifically for **Moodle** in `boxhist2`). See also Table 3 in the appendix for an overview.

3.2. The `xexams()` wrapper function

To avoid recoding certain tedious tasks – such as copying/reading files and handling temporary directories – for each of the user interfaces introduced in Section 2, the new **exams** package provides a modular and extensible framework for building new exam-generating functions. This framework is tied together by the `xexams()` function which is typically not called by users directly but forms the basis for all new `exams2`*xyz* interfaces.

To accomplish this, `xexams()` also takes the arguments listed in Table 1 (except `name` and `template`), draws exams from the exercise `file` list, and does all the necessary file/directory handling. Furthermore, it takes a `driver` argument that needs to be a list of four functions `driver = list(sweave, read, transform, write)`. These are utilized as follows:

1. *Weave:* For each of the selected exercise files (within all `n` exams) `driver$sweave(file)` is run to weave the `.Rnw` file into a `.tex` file. If `sweave = NULL` (the default), the standard `Sweave()` function is used. If `sweave = list(...)` is a list, e.g., `list(pdf = FALSE, png = TRUE)`, this is passed as arguments to `Sweave()`.

2. *Read:* Each resulting `.tex` file is read into R using `driver$read(file)`. By default (`read = NULL`), the function `read_exercise()` is used (see below), resulting in a list of character vectors with the LaTeX code for question/solution plus metainformation.

3. *Transform:* Each of these exercise-wise list objects can subsequently be transformed by `driver$transform(object)` which can be leveraged for transformations from LaTeX to HTML etc. By default (`transform = NULL`), no transformation is applied.

4. *Write:* The (possibly transformed) lists of exercises, read into R for each exam object, can be written out to one ore more files per exam in an output directory via `driver$write(object, dir, info = list(id, n))`. By default (`write = NULL`), no files are written.

After performing each of the driver functions, `xexams()` returns invisibly a nested list object (currently unclassed) as illustrated in Figure 7. It is a list of *exams* (of length `n`), each of which is a list of *exercises* (whose length depends on the length of `file` and `nsamp`), each of which is a list (whose length/contents depends on `driver$read`). When used with the default `read_exercise()`, each exercise is a list of length 6 containing the question/solution texts, metainformation, and paths to supplementary files. These will be introduced in more detail in the next section.

All of the interfaces introduced in Section 2 employ the standard `Sweave()` function for the weaving step (possibly with custom arguments) and the `read_exercise()` function for the reading step. They mainly differ in the transformation and writing step. `exams2pdf()` needs no transformation and the writer first sets up a `.tex` file for each exam, calls `texi2dvi(pdf`

```
list(                ## of 'exams', length: n
  list(              ## of 'exercises', length: k
    list(            ## of exercise content, length: 6
      question,
      questionlist,
      solution,
      solutionlist,
      metainfo,
      supplements
    )
  )
)
```

Figure 7: Structure of the return value of `xexams()`, when used with the default `read` driver `read_exercises()`.

`= TRUE)`, and then copies the resulting `.pdf` file to the output `dir`. `exams2html()` on the other hand uses a TEX-to-HTML transformation and the writer then sets up a `.html` file for each exam and copies it to the output `dir`. Finally, `exams2moodle()` and `exams2qti12()` both also use a transformation to HTML but have no writer. The reason for this is that they do not write one file per exam (i.e., with only one replication per exercise) but rather need to produce XML files that include all different replications of each exercise. Hence, they take the list returned by `xexams()` and process it subsequently in different ways. The details for all these steps are explained in the subsequent subsections.

3.3. The `read` driver: `read_exercise()` and `read_metainfo()`

The function `read_exercise()` reads the weaved exercises, i.e., files like that shown in Figure 2. It simply extracts the text lines from the `question` and `solution` environments and stores them in vectors of the same name. If these environments contain `answerlist` environments, these are extracted and stored separately in `questionlist` and `solutionlist` vectors, respectively. Finally, the metainformation is extracted using `read_metainfo()` which not only stores character vectors but also transforms them to suitable types (depending on the `extype`) and performs some sanity checks. The resulting metainformation is a list with elements essentially corresponding to the commands from Table 2.

For illustration, we run `xexams()` to select the same three exams as used in the **Moodle** and **OLAT** examples above. However, using the default `driver` specification, `xexams()` just performs the weaving and reading steps (and has no transformation or writing step):

```
R> set.seed(1090)
R> x <- xexams(myexam, n = 3)
```

The resulting object is a nested list as shown in Figure 7 with 3 exams of 5 exercises each (drawn from the `myexam` list). Using `x[[i]][[j]]`, the `j`-th exercise of the `i`-th exam can be accessed. Here, we explore the first exercise (`boxplots`, a multiple-choice question) from

the first exam that is also shown in Figures 5 and 6. Its general question text (in LaTeX) is printed below – it requires a graphic which is stored in a supplementary file in a temporary directory.

```
R> writeLines(x[[1]][[1]]$question)

In Figure~\ref{fig:ch06-boxplots} the distributions of a variable
given by two samples (A und B) are represented by parallel boxplots.
Which of the following statements are correct?  \emph{(Comment: The
statements are either about correct or clearly wrong.)}

\setkeys{Gin}{width=0.7\textwidth}
\begin{figure}[htb!]
\begin{center}
\includegraphics{boxplots-002}
\caption{\label{fig:ch06-boxplots} Parallel boxplots.}
\end{center}
\end{figure}

R> x[[1]][[1]]$supplements

                                                    boxplots-002.pdf
"/tmp/RtmpgJbOwy/file134e5dc3b5f0/exam1/exercise1/boxplots-002.pdf"
attr(,"dir")
[1] "/tmp/RtmpgJbOwy/file134e5dc3b5f0/exam1/exercise1"
```

The corresponding list of statements about the graphic is stored separately. It is shown below along with the most important metainformation elements.

```
R> x[[1]][[1]]$questionlist

[1] "The location of both distributions is about the same."
[2] "Both distributions contain no outliers."
[3] "The spread in sample A is clearly bigger than in B."
[4] "The skewness of both samples is similar."
[5] "Distribution A is about symmetric."

R> x[[1]][[1]]$metainfo[c("file", "type", "solution")]

$file
[1] "boxplots"

$type
[1] "mchoice"

$solution
[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

In summary, `xexams()` combined with the default readers is relatively straightforward to use in other progams (such as the `exams2`*xyz* functions). The return value is somewhat "raw" as it is not classed and has no dedicated methods for subsetting etc. However, we refrained from using a more elaborate structure as this function is not meant to be called by end-users while we expected other developers to find the current structure sufficiently simple to use in their programs.

3.4. LaTeX-to-HTML `transform` driver generator

When embedding statistical/mathematical exercises into web pages or learning management systems, the exercises' LaTeX text – typically containing mathematical notation – has to be transformed in some way so that it can be rendered by a browser. Until relatively recently, this posed the notorious problem of how to display the mathematical formulas and often the only good answer was to embed raster images of the formulas. However, this situation has clearly changed (see e.g., Vismor 2012) and there are now various convenient options: e.g., using the mathematical markup language MathML (W3C 2010; Wikipedia 2013) or keeping LaTeX formulas in the web page and embedding some JavaScript for rendering them.

Especially the display of MathML in web pages has become very easy: **Firefox** long had native support for it and for the Microsoft **Internet Explorer** the **MathPlayer** plugin (Design Science 2013b) has long been available. More recently, other major browsers like **Opera** or **Safari** also added support for MathML (see Vismor 2012, Section 1.2). Google **Chrome** briefly enabled MathML support but disabled it again due to instabilities. Furthermore, **MathJax** (Design Science 2013a), an open-source JavaScript engine, can be used to render MathML (or LaTeX) formulas on a server rather than in the local browser.

Therefore, the new **exams** package offers functionality for automatically transforming the LaTeX exercises to HTML within R and by default employs MathML for all mathematical notation (e.g., as demonstrated in Figure 4). More specifically, the package provides the driver generator `make_exercise_transform_html()`. It returns a function suitable for plug-in into the `transform` driver in `xexams()` which then replaces the LaTeX code in `question`/`questionlist` and `solution`/`solutionlist` with HTML code. For illustration, we set up a particular function `trafo()` below and apply it to the first exercise in the first exam within the object `x` that we had considered before:

```
R> trafo <- make_exercise_transform_html(converter = "ttm", base64 = FALSE)
R> writeLines(trafo(x[[1]][[1]])$question)

In Figure  the distributions of a variable
given by two samples (A und B) are represented by parallel boxplots.
Which of the following statements are correct?  <em>(Comment: The
statements are either about correct or clearly wrong.)</em>
<div class="p"><!----></div>
<div class="p"><!----></div>
<a name="tth_fIg1">
</a>
<div style="text-align:center"><img src="boxplots-002.png" alt="boxplots-002.png" />
<div style="text-align:center">Figure 1: <a name="fig:ch06-boxplots">
</a> Parallel boxplots.</div>
```

```
</div>
<div class="p"><!----></div>
```

It can be seen that the resulting exercise employs HTML text, e.g., uses `<em>` instead of `\emph` or `<img>` instead of `\includegraphics`.[7]

Internally, `make_exercise_transform_html()` can leverage three different `converters`: `ttm` (default), `tth`, or `tex2image`. The former two come from the R package **tth** (Hutchinson *et al.* 2013) and internally call the two C functions `tth` (TEX to HTML) and `ttm` (TEX to HTML/MathML) taken from the **TtH** suite of Hutchinson (2012). The last option, `tex2image`, is a function provided by the **exams** package itself. It proceeds by first running `texi2dvi(pdf = TRUE)` from the base R package **tools** and subsequently converting the resulting PDF to a raster image in a `system()` call to **ImageMagick**'s `convert` function (**ImageMagick** Studio LLC 2013). Thus, for this function **ImageMagick** is assumed to be installed and in the search path. All three converters have their benefits and drawbacks:

- `tth` is typically preferable if there is no or only very simple mathematical notation. The resulting HTML can then be rendered in any modern browser.
- `ttm` is preferable if there is some moderately complicated mathematical notation (e.g., fractions or equation arrays etc.). As argued above this can still be easily displayed in suitable browsers or by employing **MathJax** in the web page.
- `tex2image` is the "last resort" if neither of the two previous approaches work, e.g., if more complex LATEX commands/packages need to be used which are not supported by `tth`/`ttm`. It is fairly slow while `tth`/`ttm` are typically even faster than calling LATEX.

To explore the differences of the results, the converters can also be called directly on character strings containing LATEX. Below we use two simple code lines for which `tth()` would probably be sufficient:

```
R> tex <- c("This is \\textbf{bold} and this \\textit{italic}.",
+    "Points on the unit circle: $x^2 + y^2 = 1$.")
R> ttm(tex)

 [1] "This is <b>bold</b> and this <i>italic</i>."
 [2] "Points on the unit circle: "
 [3] "<math xmlns=\"http://www.w3.org/1998/Math/MathML\">"
 [4] "<mrow>"
 [5] "<msup><mrow><mi>x</mi></mrow><mrow><mn>2</mn></mrow>"
 [6] "</msup>"
 [7] "<mo>+</mo>"
 [8] "<msup><mrow><mi>y</mi></mrow><mrow><mn>2</mn></mrow>"
 [9] "</msup>"
[10] "<mo>=</mo><mn>1</mn></mrow></math>."
```

[7] It may be noteworthy to that the conversion (a) assumes the graphics to be in `.png` format and (b) does not resolve the figure reference at the beginning of the text correctly. For (a), we just need to make sure that the `sweave` driver in `xexams()` has `png = TRUE` (and `pdf = FALSE`) which is accounted for in `exams2html()` etc. Issue (b), however, needs to be avoided by formulating the underlying `.Rnw` differently (or by tolerating the missing number).

```
R> tth(tex)

[1] "This is <b>bold</b> and this <i>italic</i>."
[2] "Points on the unit circle: x<sup>2</sup> + y<sup>2</sup> = 1."

R> (tex2image(tex, dir = odir, show = FALSE))

[1] "/tmp/RtmpgJbOwy/file134e502205e2/tex2image_1.png"
```

Note that `tex2image(tex)` returns the path to a raster image file which by default is also shown directly in the browser.

Finally, our illustration of `make_exercise_transform_html()` also employed a second option, `base64 = FALSE`, which still deserves more detailed explanation. After converting an exercise from LaTeX to HTML code (using either of the three converters above), the HTML code may contain references to supplementary files (e.g., in `<img>` tags). Optionally, by using the default `base64 = TRUE`, these images can be embedded directly into the HTML code in Base64 encoding (via the **base64enc** package in R, Urbanek 2012) and thus waiving the need for having them as supplementary files.

3.5. PDF and HTML `write` driver generators

In the first three steps of `xexams()`, exams are randomly drawn and weaved, read into R, and potentially transformed from LaTeX to HTML (or some other format). However, so far, no output files have been generated. The original idea of Grün and Zeileis (2009) was to produce one or more output files for each of the `n` generated exams. To do so in `xexams()` a `write` driver can be specified. The package provides several generating functions for suitable drivers, especially for generating PDF and HTML files. As before, the idea is to pass customization options to the driver generator which can then be plugged into `xexams()`.

For PDF output files, the following driver generator is available:

```
make_exams_write_pdf(template = "plain", name = NULL,
  inputs = NULL, header = list(Date = Sys.Date()), quiet = TRUE,
  control = NULL)
```

This is employed in `exams2pdf()` and proceeds in the same way as described by Grün and Zeileis (2009) for the `exams()` function. It includes the `question`/`questionlist` and `solution`/`solutionlist` in a LaTeX `template`, then runs `texi2dvi(pdf = TRUE)` from the base **tools** package, and finally copies the resulting PDF files to a desired output directory. The default `plain.tex` template is provided within the **exams** package and also more than one template can be employed as illustrated in Section 2. Details about the remaining customization arguments are provided on the manual page and in Grün and Zeileis (2009).

For HTML output files, a similar driver generator is available:

```
make_exams_write_html(template = "plain", name = NULL,
  question = "<h4>Question</h4>", solution = "<h4>Solution</h4>",
  mathjax = FALSE)
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>

<head>
<title>Exam ##ID##</title>
<style type="text/css">
body{font-family:Arial;}
</style>
</head>

<body>
<h2>Exam ##ID##</h2>

##\exinput{exercises}##

</body>
</html>
```

Figure 8: Default HTML `template` file `plain.html` employed in `make_exams_write_html()`. Elements marked by `##...##` are being replaced in each replication of the exam.

This is employed in `exams2html()` and is also based on a `template`. By default the `plain.html` file is used that is provided within **exams** and shown in Figure 8. This contains placeholders marked with `##...##` that are to be replaced in each randomly drawn exam. The `##ID##` is simply replaced with a numeric ID $(1, \ldots, \texttt{n})$ and `##\exinput{exercises}##` is replaced by an ordered list (`<ol>`) containing the question/solution. If the `question` and `solution` arguments to `make_exams_write_html()` are character strings, these are added as titles in the list. Alternatively, either argument can also be set to `FALSE` which avoids inclusion of the corresponding element of the exercise in the resulting HTML file.

As an additional convenience setting `mathjax = TRUE` includes the `<script>` tag for loading the **MathJax** JavaScript. Then, **MathJax** (rather than the browser) handles the rendering of the MathML formulas (if any) in the HTML file. To experiment with this option, one can simply use examples like `exams2html("tstat", mathjax = TRUE)`.

3.6. Further functions for processing `xexams` return values

The interfaces `exams2moodle()` and `exams2qti12()` work somewhat differently compared to `exams2pdf()` and `exams2html()`. They produce a single XML file containing all `n` replications of all exercises rather than separate files per exam. The reason is that learning management systems such as **Moodle** or **OLAT** provide their own functionality for randomly drawing questions from a pool stored in the system. Hence, `exams2moodle()` and `exams2qti12()` do not really select `n` separate exams but supply a set of `n` replications (either from identical or differing templates) that can be uploaded into the systems' question pools.

Therefore, both interfaces do call `xexams()` with the standard weave/read drivers and the HTML transformer introduced above but without a write driver. Instead, the whole R list of exercise replications returned by `xexams()` is processed subsequently in one go and embedded into a suitable XML file. For the **Moodle** interface, the function has the following arguments:

```
exams2moodle(file, n = 1L, nsamp = NULL, dir,
  name = NULL, edir = NULL, tdir = NULL, sdir = NULL,
  quiet = TRUE, resolution = 100, width = 4, height = 4,
  iname = TRUE, stitle = NULL, testid = FALSE,
  num = NULL, mchoice = NULL, schoice = mchoice, string = NULL,
  cloze = NULL, zip = FALSE, points = NULL, rule = NULL, ...)
```

Thus, in addition to the usual arguments from the first two lines (see Table 1), the third line has those arguments passed to `Sweave`, and lines 4–6 have the arguments responsible for the XML formatting. These are employed in the following steps:

- A character vector with the XML code for the `<moodlequiz>` is generated.
- For each question a title text is included (in suitable XML tags), where `iname`, `stitle`, and `testid` can be used for the fine-tuning.
- The XML code for each question/exercise is inserted. It is generated by the transformation functions `num`, `mchoice`, `schoice`, `string`, and `cloze`. For example, if `x[[i]][[j]]` is a multiple-choice exercise, then `mchoice(x[[i]][[j]])` is employed to generate the XML character string.

Thus, users can supply custom functions that handle the XML question generation. By default, the package has a flexible generator `make_question_moodle23()` that returns a suitable function. Analogously to other generators employed previously, this can be easily adapted. For example, the user could set `mchoice = list(solution = FALSE, shuffle = TRUE)` and then the `mchoice` XML driver would be `make_question_moodle23(solution = FALSE, shuffle = TRUE)`. Hence, while only a single generator function is available, one can easily set different argument lists for numeric or multiple-choice exercises etc. Furthmore, other fine-control options are available, e.g., for setting the `points` for each exercise (overruling the `\expoints` metainformation) or the `rule` used for partial credits in multiple-choice exercises (see also Appendix B).

The approach take in `exams2qti12()` is essentially analogous to that of the **Moodle** interface. It also has separate `num`, `mchoice`, `schoice`, `string`, and `cloze` XML transformation functions, each of which is by default generated by `make_itembody_qti12()` (as exercises are called items in **OLAT**), possibly supplying further arguments for customization.

```
exams2qti12(file, n = 1L, nsamp = NULL, dir,
  template = "qti12", name = NULL, edir = NULL, tdir = NULL, sdir = NULL,
  quiet = TRUE, resolution = 100, width = 4, height = 4,
  num = NULL, mchoice = NULL, schoice = mchoice, string = NULL,
  cloze = NULL, duration = NULL, stitle = "Exercise", ititle = "Question",
  adescription = "Please solve the following exercises.",
  sdescription = "Please answer the following question.",
```

```
    maxattempts = 1, cutvalue = 0, solutionswitch = TRUE, zip = TRUE,
    points = NULL, eval = list(partial = TRUE, negative = FALSE), ...)

  make_itembody_qti12(rtiming = FALSE, shuffle = FALSE, rshuffle = shuffle,
    minnumber = NULL, maxnumber = NULL, defaultval = NULL, minvalue = NULL,
    maxvalue = NULL, cutvalue = NULL, enumerate = TRUE, digits = NULL,
    tolerance = is.null(digits), maxchars = 12,
    eval = list(partial = TRUE, negative = FALSE))
```

For details about the arguments see `?exams2qti12`. The main difference between the **Moodle** XML and QTI 1.2 XML specifications is that the former just provides some control over the individual exercises (or questions, items) whereas the latter also has control options for the whole exam (or assessment). Therefore, the XML specification is somewhat more complex. Hence, `exams2qti12()` also takes a `template` argument that is by default set to the `qti12.xml` file provided within **exams**. The template must contain exactly one `<section>` with exactly one `<item>` with a placeholder `##ItemBody##`. Then, `exams2qti12()` reads the template, replicates the `<section>` for each exercise, replicates the `<item>` `n` times within each `<section>`, and then inserts the `##ItemBody##` with the XML transformation functions for `num`, `mchoice`, etc.

Similar to `exams2moodle()`, one can specify the `points` for each exercise (again overruling the `\expoints` metainformation) or specify an `eval` argument that describes the evaluation policy employed (see Appendix B for details).

One detail of the QTI 1.2 interfaces should be briefly explained: Although the QTI 1.2 XML standard supports numeric exercises/items through its `<response_num>` tag, this is not fully implemented in all QTI-based learning management systems. Namely, **OLAT** has no official support for this type of exercises, although they actually work correctly.[8] Hence, optionally `exams2qti12()` offers a workaround: If the `digits` argument is some finite value (e.g., `digits = 2`), the correct numeric solution is formatted to a character string with `digits` decimal places. Then, the correct solution is entered as a string solution (`<response_str>`) which means that answers entered in the system will only be recognized as correct if they exactly match the correct string (e.g., `111.1` would not be recognized as correct if the string `111.10` with `digits = 2` is used).

In summary, most end users should just have to call the main interfaces `exams2moodle()` or `exams2qti12()` and customize by setting options for `num`, `mchoice`, etc. as some `list(...)`. If this is not sufficient, though, the users could program their own XML transformation functions for `num`, `mchoice`, etc. And finally, for QTI 1.2, a different template could be used.

4. Extending the exams toolbox and writing new drivers

In some cases it is not sufficient to use the arguments of the existing `exams2`*xyz*`()` functions or to provide alternative `template`s to them. In particular, when a completely different output format is required (e.g., a different XML format), it might be necessary to develop new drivers for the `xexams()` toolbox. One example for such a situation is the software product that is

[8] The only aspect that does not work for `<response_num>` exercises is the display of correct solutions in the final results page of an exam.

currently employed for generating *printed* large-lecture exams at Universität Innsbruck. This allows for

- specification of (static) single/multiple-choice exercises in a browser interface,
- production of so-called "scrambled" PDF exams from it (where the static questions and solutions are simply shuffled),
- optical character recognition (OCR) of scans from the exams' title pages,
- computation of the points/marks achieved by the students.

Although, the **exams** package can also generate PDF exams directly, an interface to this exam server is desirable because it can handle the OCR automatically and the students can easily log into the exam server to see their personal results and inspect their exam scan.

Fortunately, this so-called LOPS exam server (developed by a spin-off company of WU Wien) also employs an XML specification for importing/exporting its exams. Therefore, it was easily possible for us to establish a new `exams2lops()` interface that produces one ZIP file for each exam, including the XML plus supplementary graphics. A corresponding `write` driver generator `make_exams_write_lops()` is also supplied in the package. Its details are not discussed here because the XML format adopted is specific to this WU-developed software which is not widely used. The `exams2lops()` interface then essentially proceeds in the following manner:

```
htmltransform <- make_exercise_transform_html(converter = "tex2image",
  base64 = FALSE)
lopswrite <- make_exams_write_lops(...)
xexams(file, n, nsamp, driver = list(
    sweave = list(quiet = TRUE, pdf = FALSE, png = TRUE, ...),
    read = NULL,
    transform = htmltransform,
    write = lopswrite),
  ...)
```

First, an HTML `transform` driver is set up which uses the `"tex2image"` converter because the LOPS server does not support MathML. Then, it sets up the custom `write` driver using a couple of extra arguments (`...`) whose details are suppressed here for simplicity. Finally, `xexams()` is called with (1) the default `sweave` driver `Sweave()` with options set to producing PNG but not PDF graphics, (2) the default `read` driver, (3) the `tex2image`-based TEX-to-HTML `transform` driver, (4) the custom `write` driver.

Of course, the part that involves a certain amount of coding is to program the `write` driver (or driver generator, as here). However, the building blocks for the weave/read/transform steps can be easily recycled. Also, if readers of this manuscript need to code their own driver generator, we recommend to use the drivers from the **exams** package for inspiration. Last but not least, the **exams** package is hosted and R-Forge (Theußl and Zeileis 2009) and also provides a forum for support and discussions of e-learning exams in R at `http://R-Forge.R-project.org/forum/?group_id=1337`.

5. Summary and discussion

Summary

Motivated by the need for automatic generation of exams (or quizzes, tests, assessments) for learning management systems, the **exams** package is turned into an extensible toolbox for exam generation. While previous versions of the package just supported generation of random replications of exams in PDF format, the new version of the package provides interfaces for various output formats, such as PDF, HTML, or XML specifications for **Moodle** or **OLAT**. All exam output formats are based on the *same* specification of exercise `Sweave` files whose format was only slightly extended compared to previous versions. The flexibility of producing different output formats is accomplished by adopting a new extensible framework that consists of the following modular steps: (1) *weaving* a single exercise, (2) *reading* the resulting LaTeX text and metainformation into R, (3) *transforming* the text (if necessary, e.g., from LaTeX to HTML), (4) *writing* the text into output files such as LaTeX, HTML, or XML templates. Flexible building blocks are available for each of the steps that can either be customized for the existing output formats or reused for generating new output formats.

Infrastructure vs. content

As emphasized in the discussion of version 1 of **exams** (Grün and Zeileis 2009), the objective of the package is to provide the technological infrastructure for automatic generation of exams, especially for large-lecture courses. Thus, users of **exams** should not have to worry about implementation details and can focus on the content of their exams when they build up a pool of exercises accompanying a particular course. Creating "good" exercises from an educational (rather than computational) point of view is not a trivial task but guidelines for this are beyond the scope of the **exams** package and this manuscript. Hence, we just provide a few references to the relevant literature on statistical education and assessment: Gal and Garfield (1997) and Garfield and Chance (2000) discuss issues such as topics covered and skills developed in statistics courses as well as suitable ways of assessment. Strategies for good multiple-choice questions, especially if they are also used for self-study materials, are suggested by Klinke (2004).

Strategies for setting up exercises

When switching a course to the **exams** infrastructure, clearly the most work has to go into the generation of the content, i.e., the `Sweave` exercises. However, by the modular design of the package it is easy to distribute the workload among a large team of contributors. Each person can just work on stand-alone `.Rnw` files, e.g., for a particular exercises type or for the exercises pertaining to a specific chapter from the lecture etc. Depending on the output formats, it is typically a good idea to make sure that the exercise, `foo.Rnw` say, works as desired by running `exams2pdf("foo.Rnw")` and `exams2html("foo.Rnw")` to make sure that it can be appropriately rendered in both PDF and HTML. To check that the solution is correctly entered in the metainformation, it helps to run `exams_metainfo(exams2html("foo.Rnw"))` (or analogously for `exams2pdf()`).

When the pool of exercises is ready, then it is typically useful to set up a convenience wrapper function that (a) selects the desired exercises from this pool and (b) produces the desired

output format(s) for them. For the latter step, it may just be necessary to set the arguments of one of the `exams2`*xyz*`()` functions appropriately or maybe to write a custom `template` that can be plugged into the function. However, the customization of such a wrapper function is typically not a lot of work and can be performed by a single person, e.g., the team member with some more experience in the technologies involved (R, HTML, XML, ...). A useful starting point for setting up such a wrapper can be generated with the `exams_skeleton()` function, based on which different interfaces and templates can be easily explored.

Experiences at Universität Innsbruck

In 2012, the Department of Statistics at Universität Innsbruck built up infrastructure for a new "Mathematics 101" course. The team included seven professors and lecturers, and six student assistants. All professors and lecturers were previously familiar with R and LaTeX (but not necessarily with HTML or XML) while several of the student assistants had experience in neither. The workload was then split up so that the professors and lecturers designed the content of the exercises and programmed prototypes. The student assistants then typically performed tasks such as checking the correctness of the exercises, testing out the random data generation or making it more flexible, and creating variations of existing exercises by making small modifications in the underlying "stories" or changing the data generating process. Even though many of the student assistants had no prior knowledge of R and LaTeX, they were rather quickly able to work on the exercise `Sweave` files (with all the usual small problems that often occur when learning R/LaTeX).

The resulting pool of exercises is maintained in a **Subversion** repository (SVN, Pilato, Collins-Sussman, and Fitzpatrick 2004) for version control so that all team members can easily obtain the latest version or contribute fixes/improvements. In combination with the **exams** package this approach proved to be rather successful in addressing the needs of multi-author and cross-platform development.

After having the pool of exercises established, just one team member is concerned with running `exams2qti12()` and uploading the resulting ZIP file into **OLAT** for the biweekly online tests. And for creating the printed tests at the end of the semester the `exams2lops()` or `exams2pdf()` interfaces are employed.

Experiences at BOKU Wien

At BOKU a web-based online exercise system had been in place for several years (Moder 2011). The old system used Fortran to generate data and a standalone web-interface for students to enter results and get feedback. The storylines of the old system were transfered into `Sweave` files, the Fortran code was re-programmed in R for more flexibility in random number generation. Workload management was similar as in Innsbruck: Professors and lecturers supervised the effort, but programming of examples was done by a team of PhD students, two of which were new at the department and had only limited prior experience with R (but all had some experience in LaTeX).

There are three types of **exams**-generated exercises used throughout three "Statistics 101" courses totalling more than 1200 students:

- Online exercises where students get immediate feedback on correctness of results,
- homework exercises where students create a small report as PDF and upload this to the

server, and

- pen and paper multiple choice exams as in package **exams** version 1.

For the pen-and-paper multiple choice tests a mixture of `exams2pdf()` and `exams2moodle()` is currently used: Moodle in principle generates exercise and answer sheets for multiple choice exams. However, in the current implementation the question text may not contain figures and mathematical equations are really ugly. So question sheets are generated directly as PDF, but the XML for Moodle is also created, which generates then in turn the answer sheets and is used for automatic scanning and grading. A much more detailed description of contents and example types has been written as a seperate manuscript and is currently under review. The focus of this paper is to be a technical manual of the new features of package **exams**.

Outlook

In the current version, **exams** already provides a wide variety of different output formats, some additional formats may be desirable for future developments though. For example, QTI 2.0/2.1 is likely to become more widely adopted – and is already currently employed by some programs such as **ONYX**. This may also be one potential route for support of **Blackboard** which we have not yet been able to investigate due to lack of access to the proprietary **Blackboard** system. An alternative could be a direct adaptation of the **Blackboard** flavor of the QTI 1.2 XML format.

Furthermore, users may be interested in extensions/adaptions of existing e-learning formats. A forum for support and discussions of such issues is available on R-Forge at `http://R-Forge.R-project.org/forum/?group_id=1337`.

Acknowledgments

We are indebted to all our colleagues in the "Mathematics" team at the Department of Statistics at Universität Innsbruck for testing and challenging the code and making suggestions for improvement. This project was also partially supported by an e-learning grant (#2011.241) of the Universität Innsbruck.

References

Agea Á, Crespo García RM, Delgado Kloos C, Gutiérrez I, Leony D, Pardo A (2009). "Analysis of Existing Specifications and Standards for Assessment and Evaluation and Their Usage in Europe." *Deliverable D6.1*, ICOPER Network for Interoperable Content for Performance in a Competency-Driven Society. URL `http://www.icoper.org/results/deliverables/D6-1`.

Blackboard Inc (2010). ***Blackboard Learn*** *9.1*. Washington, DC. URL `http://www.blackboard.com/`.

Blesius CR, Moreno-Ger P, Neumann G, Raffenne E, Gonzalez Boticario J, Delgado Kloos C (2007). "**.LRN**: E-Learning Inside and Outside the Classroom – Supporting Collaborative Learning Communities Using a Web Application Toolkit." In B Fernández-Manjón,

JM Sánchez-Pérez, JA Gómez-Pulido, MA Vega-Rodríguez, J Bravo-Rodríguez (eds.), *Computers and Education: E-Learning, From Theory to Practice*. Springer-Verlag, Dordrecht.

BPS Bildungsportal Sachsen GmbH (2013). ***ONYX** Testsuite*. Chemnitz, Germany. URL http://onyx.bps-system.de/.

de Leeuw J (2001). "Reproducible Research: The Bottom Line." *Technical Report 2001031101*, Department of Statistics Papers, University of California, Los Angeles. URL http://repositories.cdlib.org/uclastat/papers/2001031101/.

Design Science (2013a). ***MathJax** 2.2 Documentation*. Long Beach, CA. URL http://www.MathJax.org/.

Design Science (2013b). ***MathPlayer**: Display MathML in Your Browser*. Long Beach, CA. URL http://www.dessci.com/en/products/mathplayer/.

Dougiamas M, *et al.* (2013). ***Moodle**, Version 2.5*. URL http://moodle.org/.

frentix GmbH (2013). ***OpenOLAT** 8.4 – User Manual*. Zürich, Switzerland. URL http://www.openolat.org/.

Gal I, Garfield JB (eds.) (1997). *The Assessment Challenge in Statistics Education*. IOS Press, Netherlands.

Garfield JB, Chance B (2000). "Assessment in Statistics Education: Issues and Challenges." *Mathematical Thinking and Learning*, **2**(1/2), 99–125.

Grün B, Zeileis A (2009). "Automatic Generation of Exams in R." *Journal of Statistical Software*, **29**(10), 1–14. URL http://www.jstatsoft.org/v29/i10/.

Hutchinson IH (2012). ***TtH**: A 'TEX to HTML' Translator*. C code version 4.03, URL http://hutchinson.belmont.ma.us/tth/.

Hutchinson IH, Leisch F, Zeileis A (2013). ***tth**: TEX to HTML/MathML Translators* tth/ttm. R package version 4.03-1, URL http://CRAN.R-project.org/package=tth.

ImageMagick Studio LLC (2013). ***ImageMagick**: Convert, Edit, and Compose Images*. Version 6.8.5-6, URL http://www.ImageMagick.org/.

IMS Global Learning Consortium, Inc (2002). *IMS Question & Test Interoperability: ASI XML Binding Specification Final Specification Version 1.2*. Lake Mary, FL. URL http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html.

Klinke S (2004). "Q&A – Variable Multiple Choice Exercises with Commented Answers." In J Antoch (ed.), *COMPSTAT 2004 – Proceedings in Computational Statistics*, pp. 1323–1328. Physica Verlag, Heidelberg.

Knuth DE (1984). *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts.

Knuth DE (1992). *Literate Programming*, volume 27 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, California.

Kuhn M (2013). "CRAN Task View: Reproducible Research." Version 2013-04-18, URL http://CRAN.R-project.org/view=ReproducibleResearch.

Lamport L (1994). *LaTeX: A Document Preparation System*. 2nd edition. Addison-Wesley, Reading, Massachusetts.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *COMPSTAT 2002 – Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg.

Leisch F (2012a). "Sweave FAQ." URL http://www.stat.uni-muenchen.de/~leisch/Sweave/.

Leisch F (2012b). Sweave *User Manual*. URL http://www.stat.uni-muenchen.de/~leisch/Sweave/.

Leisch F, Rossini AJ (2003). "Reproducible Statistical Research." *Chance*, **16**(2), 46–50.

Moder K (2011). "Statistical Education at the University of Natural Resources And Life Sciences in Vienna." In B Shishkov (ed.), *Proceedings of the Second International Conference on Innovative Developments in ICT (Information and Communication Technology)*, pp. 77–81.

Pilato CM, Collins-Sussman B, Fitzpatrick BW (2004). *Version Control with **Subversion***. O'Reilly. Full book available online at http://svnbook.red-bean.com/.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

RStudio Team (2013). ***RStudio**: Integrated Development for R*. **RStudio**, Inc., Boston, MA. URL http://www.RStudio.com/ide/.

Theußl S, Zeileis A (2009). "Collaborative Software Development Using R-Forge." *The R Journal*, **1**(1), 9–14. URL http://journal.R-project.org/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf.

Universität Zürich (2012). ***OLAT** 7.6 – User Manual*. IT Services, Universität Zürich, Switzerland. URL http://www.olat.org/.

Urbanek S (2012). ***base64enc**: Tools for Base64 Encoding*. R package version 0.1-1, URL http://CRAN.R-project.org/package=base64enc.

Vismor T (2012). "Viewing Mathematics on the Internet." Revision 2012-11-08, URL http://vismor.com/documents/site_implementation/viewing_mathematics/.

W3C (2010). "Mathematical Markup Language (MathML) Version 3.0." URL http://www.w3.org/TR/MathML/.

Wikipedia (2013). "MathML — Wikipedia, The Free Encyclopedia." URL http://en.wikipedia.org/wiki/MathML, accessed 2013-05-28.

A. List of Sweave exercises in exams

Table 3 lists all exercises that are currently provided as example Sweave files within the **exams** package. All of these exercises (except confint.Rnw that is not compatible with the version 2 interfaces) are copied when setting up an exams_skeleton() (see Section 2.3).

File	Type	Description
confint	num	Confidence interval for one-sample mean, i.e., result of length two (for version 1 interface; for version 2 see confint2 below).
dist	num	Very simple numeric exercise (for introductory illustrations).
lagrange	num	Lagrange optimzation under constraint. Result is randomly selected to be one of three potential quantities.
regression	num	Prediction in simple linear regression.
tstat	num	Computation of 1-sample t statistic.
anova	mchoice	Interpretation of anova() table and corresponding boxplots.
boxplots	mchoice	Interpretation of two parallel boxplots with potentially varying location, scatter, skewness, and outliers.
cholesky	mchoice	Computation of Cholesky decomposition, result is checked by randomly constructed statements about different matrix elements.
relfreq	mchoice	Interpretation of 2-way contingency table.
scatterplot	mchoice	Interpretation of scatterplot.
ttest	mchoice	Interpretation of t.test() output.
tstat2	schoice	Single-choice version constructed from the numeric tstat.
function	string	Simple string exercise asking for the name of specific R functions.
boxhist	cloze	Based on randomly generated data as (exercise-specific) .csv files some quantiles have to be computed (num) and interpretations of a boxplot and histogram have to be made (mchoice).
boxhist2	cloze	Same exercise but with custom layout of the answer fields (for exams2moodle() only).
confint2	cloze	Cloze version (for all version 2 interfaces) constructed from the numeric confint.
dist2	cloze	Extended cloze version of the numeric dist.
currency8	num	Exercise in UTF-8 encoding (with Euro and Pound symbol and German umlaut).
currency9	num	Same exercise in Latin-9 (or ISO-8859-15) encoding.
currency1	num	Same exercise in Latin-1 (or ISO-8859-1) encoding (omitting the Euro symbol).

Table 3: List of Sweave exercises provided as examples in exams/inst/exercises.

B. Evaluation policies

Evaluation of many exercise types generated by **exams** is relatively straightforward: num, schoice, and string answers can either be correct or wrong (possibly allowing for some tolerance in num answers). However, for mchoice and cloze exercises there is more flexibility: Either all parts of an answer have to be exactly correct or partial credits can be assigned.

Furthermore, even if answers can only be correct or wrong, there is an additional degree of freedom: Either wrong answers are not penalized (and thus not different from unanswered exercises) or negative points can be assigned to wrong answers. In this case, one needs to distinguish between exercises that were answered incorrectly and not attempted at all.

To conceptualize these different evaluation policies and provide some auxiliary functions for evaluating exam results within R, **exams** implements the function

```
exams_eval(partial = TRUE, negative = FALSE,
  rule = c("false2", "false", "true", "all", "none"))
```

where `partial` signals whether partial credits should be employed in `mchoice` exercises, `negative` indicates whether negative points are possible or not, and `rule` specifies the strategy for partial credits.

The function `exams_eval()` returns a list of its arguments along with several auxiliary R functions that can compare a given `answer` with the corresponding `correct` answer and assign point percentages. The details of these functions are illustrated with many examples on the corresponding manual page `?exams_eval`. The function `exams_eval()` itself will be most useful for **exams** users that obtain exam results themselves (as opposed to a learning management system), e.g., through optical character recognition or through their own custom web form. In such a situation, `exams_eval()` will provide useful building blocks for a custom evaluation policy.

More importantly, the same "vocabulary" for describing evaluation policies can be used in the `exams2qti12` and `exams2moodle` interfaces:

- **Moodle** only provides partial evaluation of multiple-choice exercises and the user can only choose how to assign partial credits. Every selected correct choice will always yield the fraction 1/#correct of points. When an incorrect choice is selected, it should lead to negative points. Five strategies are currently implemented: `"false"` uses 1/#wrong while `"false2"` uses $1/\max(\#\text{wrong}, 2)$; `"true"` uses 1/#correct (so that each wrong selection cancels one correct selection); `"all"` uses 1 (so that a single wrong selection cancels all correct selections); and `"none"` uses 0 (so that wrong selections have no effect at all). Finally, the overall points of an exercise can never become negative.

- In QTI 1.2/**OLAT**, there is some more flexibility. In principle partial credits can be switched off for multiple-choice questions and also the points assigned to an exercise can become negative.

Hence, `exams2moodle` only implements the `rule` argument while `exams2qti12` implements all three arguments (and would also allow to set them differently for different exercise types).

In both interfaces, `cloze` exercises always use partial credits without negative points for wrong answers in one of its components.

Affiliation:

Achim Zeileis, Nikolaus Umlauf
Department of Statistics
Faculty of Economics and Statistics
Universität Innsbruck
Universitätsstr. 15
6020 Innsbruck, Austria
E-mail: Achim.Zeileis@R-project.org, Nikolaus.Umlauf@uibk.ac.at
URL: http://eeecon.uibk.ac.at/~zeileis/, http://eeecon.uibk.ac.at/~umlauf/

Friedrich Leisch
Institute of Applied Statistics and Computing
Universität für Bodenkultur Wien
Peter Jordan-Str. 82
1180 Wien, Austria
E-mail: Friedrich.Leisch@R-project.org
URL: http://www.rali.boku.ac.at/friedrichleisch.html