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

This introduction to version 2 of the R/exams package is a (slightly) modified version of Zeileis, Umlauf, and Leisch (2014), published in the Journal of Statistical Software. Since its publication, the exams package has been extended in numerous ways which are not reflected in this vignette. Examples include: Exercises in R/Markdown format (instead of R/\LaTeX{}), the exams2nops interface for exams that can be automatically scanned and evaluated, export to Canvas or Blackboard etc.

For a current and more general introduction to the R/exams package see the materials at the official web page: http://www.R-exams.org/.

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 eight 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 (or 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. (2014) 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 (2014) 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/assessments 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), OLA/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 2014) 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.

The new version of the exams package for the R system for statistical computing (R Core Team 2014) 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 2014) 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

\footnote{Meanwhile, various configurations of exams in OLAT and OpenOLAT are supported. See the convenience function exams2openolat() for some more details.}
of such Sweave exercise templates. The resulting exams can be rendered into various formats including PDF, HTML, Moodle XML, or QTI 1.2² (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.

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 underlying 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 exams2xyz() 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 explained 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.

²An implementation of the newer QTI 2.1 standard is also available in the package, see ?exams2qti21. However, at the time of writing the function still needed more testing and hence was not discussed here.
## DATA GENERATION

```r
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

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

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{\text{\$Expr{Mean}} - \text{\$Expr{mu}}}{\sqrt{\frac{\text{\$Expr{Var}}}{\text{\$Expr{n}}}}} \\
& = & \text{\$Expr{tstat}}.
\end{eqnarray*}
```

The absolute value of the $t$-test statistic is thus equal to $\text{\$Expr{fmt(abs(tstat), 3)}}$.
\end{solution}

\begin{figure}

\begin{center}
\includegraphics[width=\textwidth]{tstat.Rnw}
\end{center}

\caption{A simple Sweave exercise: \texttt{tstat.Rnw}.}
\end{figure}
A machine fills milk into $200$ml packages. It is suspected that the machine is not working correctly and that the amount of milk filled differs from the setpoint $\mu_0 = 200$. A sample of $249$ packages filled by the machine are collected. The sample mean $\bar{y}$ is equal to $187.2$ and the sample variance $s^2_{n-1}$ is equal to $51.67$.

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

The $t$-test statistic is calculated by:

\begin{eqnarray*}
t & = & \frac{\bar{y} - \mu_0}{\sqrt{\frac{s^2_{n-1}}{n}}} \\
& = & \frac{187.2 - 200}{\sqrt{\frac{51.67}{249}}} \\
& = & -28.099.
\end{eqnarray*}

The absolute value of the $t$-test statistic is thus equal to $28.099$.

\begin{question}
A machine fills milk into $200$ml packages. It is suspected that the machine is not working correctly and that the amount of milk filled differs from the setpoint $\mu_0 = 200$. A sample of $249$ packages filled by the machine are collected. The sample mean $\bar{y}$ is equal to $187.2$ and the sample variance $s^2_{n-1}$ is equal to $51.67$.

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{187.2 - 200}{\sqrt{\frac{51.67}{249}}} \\
& = & -28.099.
\end{eqnarray*}
The absolute value of the $t$-test statistic is thus equal to $28.099$.
\end{solution}

%% META-INFORMATION
%% \extype{num}
%% \exsolution{28.099}
%% \exname{t statistic}
%% \extol{0.01}
1. Problem
A machine fills milk into 200ml packages. It is suspected that the machine is not working correctly and that the amount of milk filled differs from the setpoint $\mu_0 = 200$. A sample of 249 packages filled by the machine are collected. The sample mean $\bar{y}$ is equal to 187.2 and the sample variance $s^2_{n-1}$ is equal to 51.67.

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:

$$t = \frac{\bar{y} - \mu_0}{\sqrt{\frac{s^2_{n-1}}{n}}} = \frac{187.2 - 200}{\sqrt{\frac{51.67}{249}}} = -28.099.$$  

The absolute value of the $t$ test statistic is thus equal to 28.099.

---

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

---

Exam 1

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

| 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 tempfile(). |
| **sdir** | Path to the directory in which supplementary files (e.g., graphics or data files) are stored (except for exams()). Default: New tempfile(). |
| **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). |
| **verbose** | Should progress information be displayed (in version 2 interfaces only)? |

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:

```r
R> myexam <- list(
+     "boxplots",
+     c("confint", "ttest", "tstat"),
+     c("anova", "regression"),
+     "scatterplot",
+     "rlefreq")
R> odir <- tempfile()
R> set.seed(1090)
```
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 = 3` 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
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 the vignette `vignette("exams", package = "exams")` which is an updated version of Grün and Zeileis (2009). The vignette actually uses the new and somewhat more flexible `exams2pdf()` function rather than `exams()` and discusses the enhancements made for the former (see also below).

### 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()`**

As mentioned above, the function `exams2pdf()` is a more flexible reimplementation of `exams()` using the new extensible infrastructure of the `exams` package. For the user virtually nothing changes:

```r
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 `vignette("exams", package = "exams")` 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 \ref{sec:structure} explains the structure of the return values in more detail and illustrates how this can be used.\textsuperscript{3}

}\textit{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 \texttt{exams2html()} is designed analogously to \texttt{exams()}/\texttt{exams2pdf()} but produces HTML files. In case of just a single generated exam, this is displayed in a browser using base R’s \texttt{browseURL()} function\textsuperscript{4}. Again, this is particularly useful while writing/programming a new exercise template. For example,

\begin{verbatim}
R> set.seed(1090)
R> exams2html("tstat.Rnw")
\end{verbatim}

generates the HTML file shown in Figure \ref{fig:html} which corresponds directly to the PDF file from Figure \ref{fig:pdf}. 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 \ref{sec:browser}. Here, the Firefox browser is used (in Debian Linux’s rebranded Iceweasel version) 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 \texttt{tth} (Hutchinson, Leisch, and Zeileis 2013) that makes the ‘\LaTeX{}-to-HTML’ converter \texttt{TtH} (Hutchinson 2012) easily available in R. Also, by default, the \texttt{base64enc} package (Urbanek 2012) is employed for embedding graphics in Base64 encoding. More details on this approach are provided in Section \ref{sec:browser}.

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 \texttt{mydata.rda}, say, then simply including \texttt{\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 \texttt{exams2html("boxhist.Rnw")} for such an example.

Just like \texttt{exams()}/\texttt{exams2pdf()}, \texttt{exams2html()} can also generate multiple replications of randomly drawn exams via \texttt{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.

\textsuperscript{3}To obtain the same type of return value as from the \texttt{exams()} function, \texttt{exams_metainfo(exams2xyz(...))} can be used.

\textsuperscript{4}In R\texttt{Studio} (R\texttt{Studio} Team 2014), versions prior to 0.97.133, the ”\texttt{browser}” option is set to a function that cannot browse local HTML files on some platforms. Recent versions of R\texttt{Studio} have resolved this problem and \texttt{?exams2html} also provides workarounds for older R\texttt{Studio} versions.
Flexible Generation of E-Learning Exams in R

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, converter = "ttm")

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.

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.

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. 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)

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


t5Meanwhile, the default converter for Moodle is "pandoc-mathjax". Here, we use "ttm" instead which was the default at the time of writing.

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 avoid this problem. Hence, if we want to display matrices in OLAT, we generate them with extra empty columns. The cholesky.Rnw exercise template has code that can automatically do this, if enabled.
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" "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\(^7\). 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().\(^8\)

\(^7\)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.

\(^8\)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
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.) but either approach can be used optionally for all types of supplements.

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 exercise (maxattempts = Inf) but then suppress solutions completely (solutionswitch = FALSE) because otherwise the correct solution would be displayed after the first incorrect attempt.
page \texttt{exams2qt112} for details. Also, the underlying XML template can be adapted.

### 2.3. Creating the first exam

When creating the first “real” exam with \texttt{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 \texttt{exams2xyz()} 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, \texttt{exams} provides the function \texttt{exams\_skeleton()} (or equivalently \texttt{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 ‘\texttt{demo.R}’ script illustrating the use of the various \texttt{exams2xyz()} interfaces.

As an illustration, assume that we are interested in using \texttt{exams2moodle()} and \texttt{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 \texttt{exams2html()}. Hence, the code below calls \texttt{exams\_skeleton()} specifying these three writers. Here, we employ a temporary directory but users may set the \texttt{dir} argument to something like "C:/myexam/" or "~/myexam/" etc.

```r
R> dir.create(mydir <- tempfile())
R> exams_skeleton(dir = mydir, absolute = TRUE,
+     writer = c("exams2html", "exams2pdf", "exams2moodle"))
```

```r
[1] "demo-all.R" "demo-html.R" "demo-moodle.R" "demo-pdf.R"
[5] "exercises" "templates"
```

The directory then contains the files ‘\texttt{demo-all.R}’ which can be opened in any editor for R scripts – as well as separate ‘\texttt{demo-*.R}’ scripts for the different interfaces. These illustrate how to create various kinds of output using the \texttt{exams2html()}, \texttt{exams2pdf()}, and \texttt{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 \texttt{absolute = FALSE} would result in relative paths being used).

The function \texttt{exams\_skeleton()} always copies all exercise files to the directory but the ‘\texttt{demo-*.R}’ scripts only employ some selected files for each exercise type. It is easy to modify the ‘\texttt{demo-*.R}’ scripts, omitting or adding exercises that are readily available in the subdirectory.

Finally, to illustrate how different encodings can be used, \texttt{exams\_skeleton()} can also be called with an \texttt{encoding} argument, e.g., setting \texttt{encoding = "UTF-8"}. This modifies the
Flexible Generation of E-Learning Exams in R

demo-*.R scripts as well as the HTML and \LaTeX{} templates accordingly. The encodings "lat1n1" (or "ISO-8859-1") and "lat1n9" (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 exams2xyz() interfaces for generating exams in different formats (with currently \(xyz \in \{\text{pdf}, \text{html}, \text{moodle}, \text{qti12}\}\)) 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 exams2xyz() 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.

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 \texttt{\textbackslash extype\{\}} and an \texttt{\textbackslash minorsolution\{\}} and should typically also have a short \texttt{\textbackslash 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., \texttt{\textbackslash exsolution\{1.23\}}.

- **string** for questions with a (short) text answer, e.g., \texttt{\textbackslash 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., \texttt{\textbackslash exsolution\{01011\}}.

- **schoice** for single-choice questions where exactly one of the questions/statements is true and all others are false, e.g., \texttt{\textbackslash 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-
| Command       | Description                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| \text{}      | 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). |
| \textname{}  | Short name/description (to be used for printing within R).                                                                                   |
| \texttitle{} | Pretty longer title (for Moodle).                                                                                                             |
| \textsection{} | Section of the exercise (for Moodle, with slashes for subsections as in a URL).                                                               |
| \textversion{} | Version of the exercise.                                                                                                                    |
| \textsolution{} | 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., \textsolution{1.23|001|glm}. |
| \texttol{} | Tolerance for num solutions or a vector of tolerances (expanded if necessary) for cloze solutions. If unspecified the tolerance is 0.       |
| \textclozetype{} | List of types for the elements of a cloze exercise, e.g., \textclozetype{num|schoice|string} for the example above. |
| \textpoints{} | Points for (fully) correct solution. Default is 1.                                                                                          |
| \textextra[]{}{\textname} | Additional metainformation to be read and stored, e.g., for new custom interfaces. The default storage type is character, e.g., \textextra[myinfo]{1.23} yields a metainformation element myinfo of "1.23". The type can also be numeric or logical, e.g., \textextra[myinfo,logical]{FALSE|FALSE|TRUE}. |

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 \text{} and \textsolution{} are always required (but \textname{} is recommended additionally for nice printing in R).

choice answer, e.g., \textsolution{1.23|001|glm}. To specify the individual cloze types, a clozetype has to be given, e.g., \textclozetype{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 \texttt{exams} package. For illustration, we set up a multiple-choice question with three statements about Switzerland. First, we generate an \texttt{answerlist} with statements for the question.

\begin{verbatim}
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)
\end{verbatim}

\begin{itemize}
  \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{itemize}

Then the corresponding \texttt{answerlist} for the solution is set up.

\begin{verbatim}
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)
\end{verbatim}

\begin{itemize}
  \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{itemize}

For more examples see the exercise files in the \texttt{inst/exercises} directory of the \texttt{exams} source package. There are various multiple-choice questions with and without figures and/or verbatim R output (e.g., \texttt{anova}, \texttt{boxplots}, \texttt{cholesky}, among others). The files \texttt{tstat} and \texttt{tstat2} illustrate how the same type of exercise can be coded as a \texttt{num} or \texttt{schoice} question, respectively. The \texttt{cloze} type is employed in, e.g., \texttt{boxhist} or \texttt{fourfold} (with more flexible formatting specifically for \texttt{Moodle} in \texttt{boxhist2} and \texttt{fourfold2}, respectively). See also Table 3 in the appendix for an overview and Appendix C for more details on cloze exercises.

\subsection*{3.2. The \texttt{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 \texttt{exams} package provides a modular and extensible framework for building new exam-generating functions. This framework is tied together by the \texttt{xexams()} function which is typically not called by users directly but forms the basis for all new \texttt{exams2xyz} interfaces.

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

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

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

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

4. \textit{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 \texttt{driver$write(object, dir, info = list(id, n))}. By default (\texttt{write = NULL}), no files are written.  

After performing each of the driver functions, \texttt{xexams()} returns invisibly a nested list object (currently unclassed) as illustrated in Figure 7. It is a list of \texttt{exams} (of length \texttt{n}), each of which is a list of \texttt{exercises} (whose length depends on the length of \texttt{file} and \texttt{nsamp}), each of which is a list (whose length/contents depends on \texttt{driver$read}). When used with the default \texttt{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 \texttt{Sweave()} function for the weaving step (possibly with custom arguments) and the \texttt{read_exercise()} function for the reading step. They mainly differ in the transformation and writing step. \texttt{exams2pdf()} needs no transformation and the writer first sets up a .tex file for each exam, calls \texttt{texi2dvi(pdf...)}
Flexible Generation of E-Learning Exams in R

= 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 the following figure the distributions of a variable given by two samples (A and 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}
\includegraphics{boxplots-002}

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

boxplots-002.pdf
"/tmp/Rtmp75JBU/file327074a7f3e59/exam1/exercise1/boxplots-002.pdf"
attr(,"dir")
[1] "/tmp/Rtmp75JBU/file327074a7f3e59/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 right-skewed."

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

$file
[1] "boxplots"
$type
[1] "mchoice"
$solution
[1] TRUE TRUE FALSE TRUE FALSE

In summary, xexams() combined with the default readers is relatively straightforward to use in other programs (such as the exams2xyz 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. \texttt{\LaTeX}-to-HTML transform driver generator

When embedding statistical/mathematical exercises into web pages or learning management systems, the exercises’ \texttt{\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 2014; Wikipedia 2014) or keeping \texttt{\LaTeX} formulas in the web page and embedding some \texttt{JavaScript} for rendering them. Especially the display of MathML in web pages has become very easy: \texttt{Firefox} long had native support for it and for the Microsoft \texttt{Internet Explorer} the MathPlayer plugin (Design Science 2013b) has long been available. More recently, other major browsers like \texttt{Opera} or \texttt{Safari} also added support for MathML (see Vismor 2012, Section 1.2). Google \texttt{Chrome} briefly enabled MathML support but disabled it again due to instabilities. Furthermore, MathJax (Design Science 2013a), an open-source \texttt{JavaScript} engine, can be used to render MathML (or \texttt{\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
R> trafo <- make_exercise_transform_html(converter = "ttm", base64 = FALSE)
R> writeLines(trafo(x[[1]][[1]]$question)
```

In the following figure the distributions of a variable given by two samples (A and 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>

```html
<div class="p">!-~-~-~</div>
<img src="boxplots-002.png" alt="boxplots-002.png" />
<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>.\footnote{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).}

Internally, `make_exercise_transform_html()` can leverage three different converters: `ttm` (default), `tth`, or `tex2image`. The former two come from the R package `tth` \citep{hutchinson2013} and internally call the two C functions `tth` (\LaTeX\ to HTML) and `ttm` (\LaTeX\ to HTML/MathML) taken from the TtH suite of Hutchinson \citeyear{hutchinson2012}. 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 \citep{imagemagick}. Thus, for this function `ImageMagick` is assumed to be installed and in the search path. All three converters have their benefits and drawbacks:

- **ttm** is typically preferable if there is no or only very simple mathematical notation. The resulting HTML can then be rendered in any modern browser.
- **tth** is preferable if there is some standard 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
R> tex <- c("This is \textbf{bold} and this \textit{italic}.",
+ "Points on the unit circle: $x^2 + y^2 = 1$.")
R> ttm(tex)
```

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

```r
R> tth(tex)
```

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

```r
[1] "/tmp/Rtmpu75JBu/file327072ddebe0/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:
Flexible Generation of E-Learning Exams in R

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.

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)

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,...,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 `exams` 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:

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

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_moodle()` 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_moodle(solution = FALSE, ...)`.
Flexible Generation of E-Learning Exams in R

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. Furthermore, other fine-control options are available, e.g., for setting the points for each exercise (overruling the \texttt{\expoints} metainformation) or the rule used for partial credits in multiple-choice exercises (see also Appendix B).

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

\begin{verbatim}
exams2qti12(file, n = 1L, nsamp = NULL, dir = ".", verbose = FALSE, template = "qti12", name = NULL, edir = NULL, tdir = NULL, sdir = NULL, quiet = TRUE, resolution = 100, width = 4, height = 4, encoding = ",", 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), fix_num = TRUE)
\end{verbatim}

For details about the arguments see \texttt{?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, \texttt{exams2qti12()} also takes a template argument that is by default set to the \texttt{qti12.xml} file provided within \texttt{exams}. The template must contain exactly one \texttt{<section>} with exactly one \texttt{<item>} with a placeholder \texttt{##ItemBody##}. Then, \texttt{exams2qti12()} reads the template, replicates the \texttt{<section>} for each exercise, replicates the \texttt{<item>} n times within each \texttt{<section>}, and then inserts the \texttt{##ItemBody##} with the XML transformation functions for \texttt{num}, \texttt{mchoice}, etc.

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

One notable feature of the QTI 1.2 interfaces should be briefly explained: The fix_num argument is provided to work around an error in OLAT. While numeric exercises/items are not officially supported, they actually work correctly except for the display of the correct answer in the final summary. This is fixed (by default) by extending the processing of the answer (without changing the result). Another optional route for numeric exercises is to process them up to a certain number of digits only (e.g., \texttt{digits = 2}) and store them as a string. However, in that case, the tolerance cannot be employed.
In summary, most end users should just have to call the main interfaces \texttt{exams2moodle()} or \texttt{exams2qti12()} and customize by setting options for \texttt{num, mchoice, etc.} as some \texttt{list(...)}. If this is not sufficient, though, the users could program their own XML transformation functions for \texttt{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 \texttt{exams2xyz()} functions or to provide alternative templates 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 \texttt{xexams()} toolbox. One example for such a situation is the software product that is currently employed for generating \textit{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 \texttt{exams} package can also generate PDF exams directly, some courses still prefer the interface that have been used previously.

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 \texttt{exams2lops()} interface that produces one ZIP file for each exam, including the XML plus supplementary graphics. A corresponding \texttt{write} driver generator \texttt{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 \texttt{exams2lops()} interface then essentially proceeds in the following manner:

\begin{verbatim}
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),
...)
\end{verbatim}

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 (\ldots) whose details are suppressed here for simplicity. Finally, \texttt{xexams()} is called with (1) the default \texttt{sweave} driver \texttt{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 \url{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 \texttt{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 \texttt{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 (Grünn 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 \texttt{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 \texttt{.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, \texttt{foo.Rnw} say, works as desired by running \texttt{exams2pdf("foo.Rnw")} and \texttt{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 \texttt{exams_metainfo(exams2html("foo.Rnw"))} (or analogously for \texttt{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 \texttt{exams2xyz()} 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 (\texttt{R}, HTML, XML, ...). A useful starting point for setting up such a wrapper can be generated with the \texttt{exams_skeleton()} function, based on which different interfaces and templates can be easily explored.

\textit{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 \texttt{R} and \texttt{\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 \texttt{R} and \texttt{\LaTeX}, they were rather quickly able to work on the exercise \texttt{Sweave} files (with all the usual small problems that often occur when learning \texttt{R}/\texttt{\LaTeX}).

The resulting pool of exercises is maintained in a \texttt{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 \texttt{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 \texttt{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 \texttt{exams2lops()} or \texttt{exams2pdf()} interfaces are employed.\textsuperscript{10}

\textit{Experiences at BOKU Wien}

At BOKU a web-based online exercise system had been in place for several years (Moder \textsuperscript{10}Meanwhile, further OCR infrastructure for scanning multiple/single-choice exams within \texttt{R} has been developed, see \texttt{exams2nops()}.
Flexible Generation of E-Learning Exams in R

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 transferred 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 separate 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. The current version of exams already provides a new exams2qti21() function which so far has only been tested with ONYX. Hence, the function will probably be changed and improved somewhat in future versions of the package.

Furthermore, we have started to explore support for the proprietary Blackboard system. While this is, in principle, based on QTI 1.2, Blackboard employs its own and somewhat special flavor of QTI 1.2.

Finally, 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. Furthermore, code patches, discussions, and feedback by Dominik Ernst (Universität für Bodenkultur Wien) and Niels Smits (Vrije Universiteit Amsterdam) are gratefully
acknowledged. 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 (2014). ONYX Testsuite. Chemnitz, Germany. URL http://onyx.bps-system.de/.

dee 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 2.3 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. (2014). Moodle, Version 2.6. URL http://moodle.org/.

frentix GmbH (2014). OpenOLAT 9.4 – 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.

Grünn B, Zeileis A (2009). “Automatic Generation of Exams in R.” Journal of Statistical Software, 29(10), 1–14. doi:10.18637/jss.v029.i10.

Hutchinson IH (2012). TiH: A ‘TeX to HTML’ Translator. C code version 4.03, URL http://hutchinson.belmont.ma.us/tth/.
Flexible Generation of E-Learning Exams in R

Hutchinson IH, Leisch F, Zeileis A (2013). tth: \TeX to HTML/MathML Translators tth/ttm. R package version 4.3-1, URL http://CRAN.R-project.org/package=tth.

ImageMagick Studio LLC (2014). ImageMagick: Convert, Edit, and Compose Images. Version 6.8.9-1, 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 (2014). “CRAN Task View: Reproducible Research.” Version 2014-01-20, 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 (2014). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

RStudio Team (2014). 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 (2014). “Mathematical Markup Language (MathML) Version 3.0 2nd Edition.” URL http://www.w3.org/TR/MathML/.

Wikipedia (2014). “MathML — Wikipedia, The Free Encyclopedia.” URL http://en.wikipedia.org/wiki/MathML, accessed 2014-05-29.

Zeileis A, Umlauf N, Leisch F (2014). “Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond.” *Journal of Statistical Software*, 58(1), 1–36. doi:10.18637/jss.v058.i01.
### 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 optimization 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 exercise confint.       |
| dist2    | cloze | Extended cloze version of the numeric dist.                                                       |
| fourfold | cloze | Computation of joint probabilities in a fourfold table based on given (conditional) probabilities. |
| fourfold2 | cloze | Same exercise but with table layout of the answer fields (for exams2moodle() only).             |
| 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 do not differ 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

```r
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(#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.
C. Cloze exercises

Many large-scale exams focus on multiple- or single-choice exercise because these are easy to evaluate, especially when conducted in classical printed form. However, e-learning exams offer many more possibilities beyond the multiple/single-choice format by combining different types of questions in a single exercise. The exams package provides cloze exercises that can combine numeric, multiple/single-choice, and string questions, see Section 3.1 for an overview. As this type of exercises is very flexible, it can be both, very useful especially for statistics exercises but also somewhat more challenging to set up. Hence, we provide some additional explanations for cloze exercises here.

Figure 9 shows the boxhist exercise (see also Table 3) as a typical cloze in OpenOLAT. It was produced by

```r
R> set.seed(1090)
R> exams2qti12("boxhist", n = 1, name = "boxhist")
```

![Figure 9: Display of boxhist exercise in OpenOLAT (as rendered by a Firefox browser).](image)
It provides the link to a data file with comma-separated values (CSV) that can be downloaded by the test takers and analyzed. There are six questions: three single-choice questions (about modality, skewness, and outliers) and three numeric questions about quartiles. In Moodle the display is similar but drop-down menus are used for the single-choice questions.

To set up such a cloze exercise, separate solutions have to be provided for all questions, in this particular random version as \texttt{\textbackslash exsolution\{10\mid 001\mid 10\mid 2.54\mid 3.9\mid 3.72\}} along with the corresponding types as \texttt{\textbackslash exclozetype\{schoice\mid schoice\mid schoice\mid num\mid num\mid num\}}. The individual questions shown in the exercise are taken from the \texttt{answerlist} within the \texttt{question} which has to consist of 10 (= 2+3+2+1+1+1) items. Hence, if a corresponding \texttt{answerlist} is also provided in the \texttt{solution}, it also has to consist of 10 items. (Alternatively, the \texttt{solution} can also have no \texttt{answerlist} at all.) See the \texttt{boxhist} (or the \texttt{fourfold}) exercise in the \texttt{exams} package for examples how to dynamically produce all metainformation commands in the correct format.

Furthermore, the Moodle XML format allows for finer control of the arrangement of the answer fields in the question. By default, all questions are collected at the end of the exercise.

Figure 10: Display of \texttt{fourfold} (top) and \texttt{fourfold2} (bottom) exercise in Moodle (as rendered by a Firefox browser).
Flexible Generation of E-Learning Exams in R

(based on the corresponding `answerlist`). Alternatively, the answer fields for the individual questions can be placed anywhere within the text, arranged in a particular layout (e.g., a matrix or a table) etc. To do so, one simply has to place fields `##ANSWER1##`, `##ANSWER2##`, ... within the `question` text. Then, `exams2moodle()` will replace the placeholders by the corresponding answer fields (i.e., drop-down menus or numeric entry fields). The `boxhist2` exercise in `exams` shows how to do this as an alternative to `boxhist`.

Another illustration for a cloze exercise is given by `fourfold` and `fourfold2`, respectively.

The exercise provides some (conditional) probabilities and asks about the corresponding joint probabilities in the associated fourfold table. In `fourfold` (see the upper panel of Figure 10) the standard layout as a list of question fields is used while `fourfold2` employs the custom Moodle layout (see the lower panel of Figure 10), actually displaying a fourfold table (plus marginal sums). Both versions have again been created after `set.seed(1090)`. Users of the `exams` infrastructure thus have to decide whether they need more flexibility through setting up clozes that can be processed by all `exams2xyz()` interfaces (such as `boxhist` or `fourfold`) or whether they prefer to have a nicer layout (as in `boxhist2` or `fourfold2`). If the latter is chosen, then we recommend to always include a `answerlist` in the `question` even in cases where it is not necessary (as in `fourfold2`) so that the exercise can be processed without error in `exams2html()` or `exams2pdf()` when programming/checking the exercise.

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: https://www.zeileis.org/, http://eeecon.uibk.ac.at/~umlauf/

Friedrich Leisch
Institute of Applied Statistics and Computing
Universität für Bodenkultur Wien
Peter Jordan-Str. 82
1190 Wien, Austria
E-mail: Friedrich.Leisch@R-project.org
URL: http://www.rali.boku.ac.at/leisch.html