LaTeX Tools for Web Publishing, Screen Presentations, and Electronic Examinations

Miroslav Ćirić*

Abstract – **TeX is generally considered to be the best way to typeset complex mathematical formulas, but, especially in the form of LaTeX and other template packages, is now also being used for many other typesetting tasks. The purpose of this paper is to present the usage of TeX in preparation of high-quality PDF documents for web publishing, screen presentations and electronic exercises and examinations.**

Keywords – **TeX, LaTeX, PDF, AcroTeX.**

1. A Brief Introduction to TeX

Most people place text on a computer and arrange material on a page with a word processor. Word processors are easy to begin with. To get a blank line between two paragraphs we enter it in, to make a reference to the bibliography we type it into the text in style that we need, etc. Seems simple. We know what we want and we just do it. But, a word processor will suit our needs only if our documents are brief, short to medium sized, structurally simple, and entered by hand. As the document gets to be a bigger and tougher job, laying it out ourselves becomes a problem. For example, in a document with hundreds of bibliographic entries we can not be sure that all the entries are formatted in the same way. Even bigger problem appears when you have to typeset technical material containing a lot of mathematical symbols and complex mathematical formulas. In such cases, the best solution is to use TeX.

TeX is a system for computer typesetting created by Donald Knuth in 1978 (see [6]). It sets normal text beautifully, with line breaking algorithms that are noticeably better than those used by common word processors, but it really excels when it comes to setting extremely complicated material, containing a lot of mathematical symbols and complex mathematical formulas, such as is common in science, engineering and mathematics. TeX automatically classifies each mathematical symbol as a variable, or a relation, etc., and sets them with appropriate amounts of surrounding space. It also sizes subscripts, superscripts, and many other things. Being written by one of the world's leading experts in the design of algorithms, TeX possesses more sophisticated algorithms for making paragraphs and for hyphenating than those used by common word processors, it is fast and less memory consuming.

There are also many other advantages of TeX. It arose in the world of science and engineering where there is a tradition of cooperating closely with fellow workers. A binary input format, especially a proprietary one, is bad for these purposes, due to problems with compatibility and portability from one to another computing platform, the fact that binary files are usually big and not easy for transfer, etc. On the other hand, the TeX's input is a plain text so TeX's source files are very compact, relatively small and easy to transfer, and are portable to any computing platform. They are also easy to produce automatically, for example, when you want to write a report from material in a database. There are even ways to run TeX directly from XML input, which many people think is the standard output format of the future. TeX runs anywhere, on Windows, Macintosh, Unix (Linux) or almost any other system. It is a free and open software, the license of TeX allows free distribution and modification but demands that any changed version not be called TEX, TeX, or anything confusingly similar, providing rights similar to those of a trademark.

The TeX output can be anything. TeX's results can be converted to a printer language such as PostScript, a web language such as PDF or HTML, or, probably, to whatever that will appear in the future. And, the typesetting (line breaks, etc.) will be the same no matter where our output appears. TeX became the standard, and many publishers of technical material are set up to work with it. Most computer algebra systems, such as Maple and Mathematica, give output in TeX. And no doubt technical software developed in the future will support TeX, too.

TeX is stable, but not rigid. It is extendible, innovations can be easily added on. The most significant extension of TeX is LaTeX, whose major features include a strong focus on document structure and the logical markup of text, automatic numbering and cross-referencing, and much more. LaTeX was originally written in 1984 by Leslie Lamport (see [8]) and has become the dominant method for using TeX. The current version is LaTeX2ε developed by the LaTeX3 team[1].

*Department of Mathematics and Informatics, Faculty of Sciences and Mathematics, Višegradska 33, P. O. Box 224, 18000 Niš, Serbia and Montenegro, E-mail: ciricm@bankerinter.net

[1] http://www.latex-project.org/latex3.html

LaTeX is based on the idea that authors should be able to concentrate on writing within the logical structure of their document, rather than spending their time on the details of formatting. It encourages the separation of formatting from content, whilst still allowing manual typesetting adjustments where needed. By keeping the formatting details in a separate file from the text and making the type style, size, and vertical spacing uniform throughout the document, LaTeX is often regarded as much superior to word processors and most other desktop publishing systems. LaTeX also provides great flexibility in formatting while maintaining the identity of structure, which purely structural systems like SGML and XML do not directly address. LaTeX is itself extendible. It can be arbitrarily extended by using the underlying macro language for developing custom formats, so there are a thousands of "style files" which do everything from adapting the basics to the needs of American Mathematical Society, to making cross-references into hyper-references, etc.

Another significant extension of TeX is AMS-TeX, produced by the American Mathematical Society (AMS) to meet the standards of the AMS for publication. It provides many additional mathematical constructs and fonts with many more mathematical symbols than the fonts that come with TeX. AMS-LaTeX is a common extension of LaTeX and AMS-TeX obtained by combining the features of AMS-TeX with the ones of LaTeX. It provides all of the functionality of LaTeX (as its extension) and the functionality of AMS-TeX in LaTeX syntax and access to additional mathematical constructs and mathematical symbols not present in LaTeX. In LaTeX 2ε this is achieved in a simpler way, using amsmath, amsfonts and amscls document class packages.

2. DVI, PostScript and PDF

The TeX system has precise knowledge of the sizes of all characters and symbols, and using this information, it computes the optimal arrangement of letters per line and lines per page. It then produces a DVI file (for "device independent"), as its primary output format, containing the final locations of all characters. This DVI file can be printed directly given an appropriate printer driver or viewed on the screen using some of many existing viewers. DVI was intentionally designed to be as compact as possible and uses variable-length data structures for maximal efficiency. With many years of experience of interpreting DVI, authors of DVI previewers have achieved extremely high rates of interpretation and display, and some combine high-speed interpretation and display with very effective anti-aliasing/grey-scaling. But, the widespread use of exotic fonts, and use of TeX's `\special` primitive to convey additional information to the DVI driver, has resulted in a device-independent format which all too often has device-dependent data embedded within it. As a result, DVI is acceptable only within the narrow TeX community, and for portability DVI files have to be converted to another format before being sent to those outside of the TeX community. PostScript is supposed to be this format, but Portable Document Format (PDF) turned out to be an even better candidate for such purposes.

PDF is a file format developed by Adobe Systems for representing documents in a manner that is independent of the original application software, hardware, and operating system used to create those documents. A PDF file can describe documents containing any combination of text, graphics, and images in a device independent and resolution independent format. PDF is derived directly from Adobe's PostScript language, and it is primarily the combination of three technologies: (1) a cut-down form of PostScript for generating the layout and graphics, (2) a font-embedding/replacement system to allow fonts to travel with the documents, and (3) a structured storage system to bundle these elements into a single file, with data compression where appropriate. Whilst PostScript is intended primarily as a page-description language which will normally be interpreted within dedicated printer logic, PDF is aimed far more at on-screen display, document exchange, and hypertextual applications. A PDF document is usually smaller than the equivalent PostScript document, has better-defined structure, allows both intra- and inter-document links, various dynamic effects for page transitions and working with forms, but lacks the procedural elements which allow a PostScript file to perform (sometimes quite significant) computation within the printer engine.

As well as defining the language/format PDF, Adobe have also produced a suite of tools for creating, viewing, printing, indexing, searching and modifying PDF documents. This suite, called Adobe Acrobat, is available for a wide range of platforms, and the Acrobat Reader is in fact available entirely free of charge. Acrobat Reader has a font-substitution strategy that ensures the document will be readable even if the end-user does not have the "proper" fonts installed, and it integrates perfectly with modern web browsers, which allows both local and remote PDF documents to be viewed within the same window as analogous HTML documents. Modern web search engines search, classify and catalogue PDF documents in an entirely transparent manner.

There are several methods for producing PDF documents when the source document is written in TeX or LaTeX. The first one consists of the three steps: (1) compiling TeX source file to DVI file which contains `\special` commands for PDF support, (2) converting DVI file to PostScript by some DVI-to-PostScript driver, such as dvips or dvipsone, and (3) translating PostScript file to PDF by some PostScript-to-PDF translator, such as Acrobat Distiller or Ghostscript. Another way is to use some of DVI-to-PDF drivers – dvipdf made by Sergey Lesenko, or dvipdfm by Mark A. Wicks, which can simplify this process by eliminating the need for PostScript generation. And the third way is to use pdfTeX or pdfLaTeX created by Hàn Thê Thành, which can produce PDF output directly from

a TEX or LATEX source, without generating DVI.

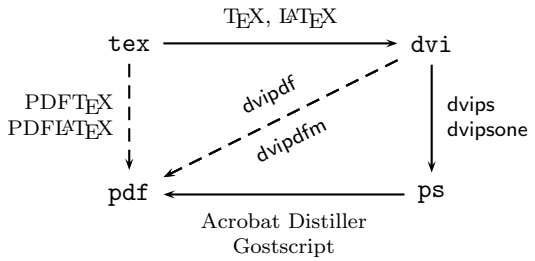

Fig 1. From TEX to PDF

Figure 1 illustrates the ways of creating a PDF document from a TEX source.

3. Web Publishing

Many people in the scientific community use PDF as a way of distributing their own works, scientific papers or lecture notes, over the Web. The material is converted from the format of the authoring application to PDF and uploaded to the Internet. Interested individuals who want to review the material using the Acrobat Reader can be assured that they will see the document as originally designed by the author. These documents, however, are simply an electronic image of the printed work, they usually contain no color and no interactive elements. In this context, it is really the author's intention that people should download and print the document to read it. PDF, however, is capable of much more than this very straight forward, yet important use.

Any publication needs an audience and quality writing to be successful. Additionally, a technical tutorial has to have several attributes to be successful on the Web: a good screen design, an attractive use of color, usage of hypertext links for cross-referencing etc. True electronic materials are meant to be read on-screen, not printed and then read, but it is very tiring to read from a computer monitor for long periods of time, so it is very important to have good screen design. There are three major points concerning screen design: (1) design the text region so that a single page fits on a screen monitor, (2) make the dimensions of the page roughly 3 by 2 (width by height), and (3) crop the pages to trim off all unnecessary white space around the margins. Having the whole page fit on the screen allows the reader to avoid constant vertical scrolling, that can be distracting and fatiguing when reading large amounts of material. Rather than scrolling, it is much easier to simply paginate, go to the next page to continue reading. The dimensions of 3 by 2 and cropping of white space from around the page will allow the user to magnify the page and the font size to help the eyes read a large amount of text on a screen for long periods of time.

To enhance the look of the document, we may occasionally include a little color. We may add color to a particular word or phrase, put a colored frame or box around an important point, paint the background a color other than the usual white. Color support in LATEX is provided through David Carlisle's color package. The command `\pagecolor`, defined by the color package, sets the background color for the current and following pages, `\color` is a declaration to switch to setting text and other objects in the given color, `\textcolor` sets the text of its argument in the specified color, `\colorbox` sets its argument in a box with the given color as a background, and `\fcolorbox` is like `\colorbox`, with a frame of the first specified color around a box with the second specified color as a background. Colors are specified either by a defined name, or by the form `[model]{specs}`. The color package supports the `rgb` (red, green, blue), `cmyk` (cyan, magenta, yellow, black), `gray`, and `named` models of color. The `named` model accesses colors by internal names that were originally built into the dvips driver, but which may be used by some other drivers, too. Note that background effects can be also created using raw postscript commands (with `\special` command).

For LATEX users, the easiest way of acquiring hypertext links (and also form features) in their Web publications is to use the hyperref package written by Sebastian Rahtz and later maintained by Heiko Oberdiek. This package extends the functionality of all the LATEX cross-referencing commands (including the table of contents, bibliographies etc) to produce `\special` commands which a driver can turn into hypertext links. It also provides new commands to allow the user to write ad hoc hypertext links, including those to external documents and URLs. When using the hyperref package, the standard LATEX labels are turned into destination pdfmarks and the cross-references are made into hypertext links understood by the Acrobat Reader. A marker being a target of a hypertext jump can be created using either the `\label` command from the standard labeling system of LATEX, which can appear after a section, subsection, equation, figure, or an enumerated item, or the `\hypertarget` command from the hyperref package, which can be used in other situations. The hyperref package provides all kinds of jumps (within the document, to another document or to a full URL), automatically adds bookmark code to an auxiliary file for `\sections`, `\subsections`, etc. Nice looking icon buttons linked to jumps can be created either using `\colorbox` or `\fcolorbox` from the color package, or using the standard LATEX graphics packages to include `eps` pictures. A description of hyperref and all its options may also be found in [5].

A Web document really needs more than simply hypertext links to cross-reference concepts, it needs user participation to get them involved with the material. This can be achieved by bringing in Acrobat Form elements and JavaScript. The hyperref package and the AcroTEX eDucation Bundle, that will be discussed in details in Section 5, define commands for creating all types of Acrobat Form elements except signature fields. They provide support to *button fields* – push-buttons, check boxes and radio-buttons, *choice fields* – list boxes, pop-up boxes and combo boxes, and *text fields* – text boxes and multiline

text fields. A form field may simply gather data from the user, and additionally, it may perform one or more actions. Actions include execute JavaScript code, going to a particular page in a document, open a file, execute a menu item, reset a form, play media or a sound, and so on. JavaScript can be attached to a PDF document in many ways, for example, to a form button. Any JavaScripts that are repeatedly used, are general (not form specific), or are rather lengthy, can be placed at what Acrobat calls the document-level JavaScripts (DLJS). A button action, then, can simply call these DLJS to perform various calculations or tasks. For inserting simple JavaScripts, AcroTEX defines the command `\JS`, and for more complex or lengthy JavaScripts, additional package insdljs is provided. It enables complex JavaScripts to be written right in the LaTeX source file, and then inserted into the section of the PDF document where the document-level JavaScripts reside. When the document is LaTeXed, the script is written verbatim to an FDF (Forms Data Format) file. The AcroTEX also adds an open action, so that when the newly created PDF document is opened for the first time in Acrobat, the FDF file is imported and executed. After the JavaScript has executed, the next thing to do is to save the document. The FDF that is imported is not saved with the document, and will not be imported again into the document, thereafter. The document is then ready for distribution. The document author can create the PDF document using the Acrobat Distiller, pdftex or dvipdfm, and the Acrobat Viewer (not the Acrobat Reader) is needed to import and execute the JavaScript.

To create a quality PDF document from a TEX source it is also necessary to use Type 1 fonts. The traditional font used by many freeware TEX systems is the bitmap or pk font. These fonts look choppy and jagged when incorporated into a PDF document and viewed on screen (though they do print decently). But, quality Type 1 Computer Modern fonts have been made available by a consortium of Bluesky Research, Y&Y, AMS, SIAM, IBM, and Elsevier. The freeware, shareware, and commercial TEX systems now come with Type 1 fonts. For an author wanting to publish on the Web using PDF, every effort must be made to reconfigure their TEX system to use these quality fonts.

4. Screen Presentations

Preparing a presentation usually means creating some sort of slides. The more multimedia projectors get common in working environments, the more comes to mind creating such presentation material as a screen version, which can be viewed using a multimedia projector or at least a computer screen. As a side effect such presentations can usually easily be presented on a web site.

Most people usually prepare their presentations using Microsoft's PowerPoint, a commercial software product to be used on a PC with the Windows operating systems, or MagicPoint, a free software for Unix. PowerPoint, for example, offers a wide range of special effects that can be selected from menus. It is easy to create backgrounds, include graphics, prepare animations, to fade in and out parts of the text dynamically, to highlight parts depending on the current state of the presentation, etc. However, scientific texts require an easy and efficient inclusion of specialized symbols and mathematical formulas. Both PowerPoint and MagicPoint cannot help with that. If there are only a very small number of formulas in a text, one may be able to include them as special graphics. But in a mathematically oriented text this strategy will be much too complicated. In recent years, a few LaTeX packages and supporting utilities have been developed to help create scientific presentations. The slides are prepared in LaTeX, then converted to PDF and the slides are presented using Acrobat Reader.

Using PDF has proved to be popular due to its ability to present the highest quality of mathematical typesetting, as well as its ability to perform a number of "transition" and other effects that can be used to give a snappy, professional-looking presentation. In order to make the process of replacing a page with the next more attractive, PDF enables to choose among several page transitions: `Split` – two lines sweep across the screen to reveal the new page similar to opening a curtain, `Blinds` – similar to `Split`, but with several lines resembling "venetian blinds", `Box` – a box enlarges from the center of the old page to reveal the new one, `Wipe` – a single line "wipes" across the old page to reveal the new one, `Dissolve` – the old page "dissolves" to reveal the new one, `Glitter` – similar to `Dissolve`, except the effect sweeps from one edge to another, and `Replace` – the old page is simply replaced with the new one without any special effect (this is the default). For some of the transitions additional parameters may be given to specify duration of the transition effect in seconds, direction of the movement, whether the effect is performed from the center out or the edges in, etc. The page transitions are always activated when opening the page, irrespective of the previous page. Therefore it does not matter whether the page is displayed through manual navigation, by page number, or a link. Generally, a page transition will be defined for the current page, but, it is also possible to define transitions for another page. A PDF document may also specify several viewer preferences which apply when opening the file in Acrobat Reader. Given suitable hardware and software, Acrobat is capable of playing sound and video files. External sound files and video clips both are stored with the `/Movie` key in the PDF. Both types of data are treated as external data. Actually, embedding the movie or sound data in the PDF file is not possible, only linking. Not all file formats are supported on all platforms. A list of supported sound and video formats can be found in the Acrobat documentation. A way for playing sound and video files, as well as for launching external programs, depend heavily on the operating system platform on which the document

is viewed, so the document is no longer portable.

A live presentation also requires ability to uncover a page step by step. It may be better in some cases, when a reader can read ahead and catch the overall view in advance, but, on the other hand, one may have an unexpected or surprising development. If this is within the range of the current slide, the remarkable item should neither go on the next slide nor be visible from the beginning. Doing a presentation with Acrobat Reader one can give the effect of dynamically building a page, because pages are updated instantaneously. If one wishes to uncover a page in several steps, one can make a sequence of pages and add some more text on each of them. The only item to keep in mind that one has to avoid updating the page or slide number between the intermediate pages, if one has numbers for general orientation or reference.

There is now a variety of specialized LaTeX packages that can aid the development of presentations in PDF format rich in color and special effects. An exhaustive overview of most of these packages is given in a paper by M. Wiedmann [14], and here we will mention only the most popular among them. Generally, all these packages can be divided into two classes. *Slide Development Packages* are LaTeX document classes and other accessories which define PDF specials producing PDF presentations with various dynamic effects: background colors and gradients, transitions effects and step by step presentation of talking points. *Slide Enhancement Tools* are either programs which are used to post-process presentations in PostScript or PDF format made by other slide development packages, in order to provide some additional special effects, or are add-ons to other document classes which create special effects. Most of the slide development packages require color and hyperref packages, to provide access to color, hyperlinks and other navigation tools.

Nice looking PDF documents can be prepared by means of pdfscreen and pdfslide packages implemented by C. V. Radhakrishnan. They are used for online readable documents (pdfscreen) and slide presentations (pdfscreen). These packages are quite popular among the pdfLaTeX users, who prefer to create a PDF output directly from a LaTeX source. A characteristic of these packages is the navigation panel that can be customized and positioned at user's will either to the left side or right side, an idea which have been also followed by many other packages. The background of the screen area can be overlayed with a graphic file, and in the case of pdfscreen, alternatively, a background color can be specified. Neither pdfscreen nor pdfslide enable to uncover a page step by step, but this can be achieved by post-processing presentations made by these packages.

The most known post-processor is PPower4 (PDF Presentation Post Processor) developed by Klaus Guntermann. The free post-processing software is written in Java and it has been run successfully with Java 1.2.x and better. PPower4 provides a small LaTeX style file (pause.sty) which let's the user insert small colored spots (using the command `\pause`) in the PDF file where a break should be make during display. During postprocessing PPower4 removes these colored chunks and adjusts the page number. This makes an impression that the same page is displayed step by step. Additional style files are provided for setting background colors (background.sty) and for simplifying access to page transitions (pagetrans.tex). Another slide enhancement tool that works well in conjunction with pdfslide and pdfscreen, as well as with many other slide development packages, is TeXPower, written by Stephan Lehmke. It is a bundle of style and class files used as an add-on to other document classes which just adds dynamic presentation effects and some other gimmicks specifically interesting for dynamic presentations. TeXPower is meant as an alternative to PPower4 for those who can not use pdfLaTeX (for instance, for PSTricks users), because it can also be used with `tex` → `dvi` → `ps` → `pdf` workflow. No post-processing or additional tools are needed - the standard LaTeX distribution will do. However, using pp4slide.sty with TeXPower and then postprocessing the PDF presentation with PPower4, some additional effects can be achieved.

One of the most complete slide development packages is Seminar, implemented by Timothy van Zandt, and maintained by Denis Girou. Among the others, it offers a variety of background types (solid, gradient and composite backgrounds, backgrounds with external images), all the transition effects supported by PDF, overlays, which allow to compose animated graphics and may be used to display a slide in several steps (cumulative overlays), and for making appear and disappear some elements on a slide (progressive overlays), various navigation bars and panels, the possibility to launch external applications, play a sound or show external movies inside a presentation, etc. Almost all of these features are is based on the PSTricks – a powerful LaTeX package which enables to use the major part of PostScript graphics and drawing capabilities inside LaTeX. Being based on PSTricks, the Seminar package needs a `tex` → `dvi` → `ps` → `pdf` workflow. There is also a lot of other slide development packages based on Seminar, as, for example, Prosper.

Note that many of the mentioned packages are able to generate both the screen and the paper version in a different output format (PDF and PostScript ones)

5. Electronic Exercises and Quizzes

In recent years, many systems have been developed for electronic testing in various areas. Most of them are based on HTML, but in areas in which a lot of mathematical symbols and formulas is used, LaTeX based systems with PDF output have shown oneself to be a better solution. In this section we present two such systems.

The AcroTeX eDucation Bundle is a collection of LaTeX macro packages for creating online exercises and quizzes in the Portable Document Format. It consists of the web

package, used to create an eye pleasing page layout suitable for the Web, classroom or conference presentations, the exerquiz package, for creating interactive exercises and quizzes, the insdljs package, which allows for the automatics insertion of document-level JavaScript, and the dljslib package, used as a library of JavaScript functions.

The exerquiz package defines three main environments: exercise, shortquiz and quiz. The exercise environment, together with a solution environment nested inside it, provides macros for creating online exercises. With these environments, we can create questions (exercises) with solutions. Solutions are written to an auxiliary file, then input back in near the end of the document, and a hypertext link is created to connect the exercise with the solution. An exercise with multiple parts can also be defined, with hypertext links to the solutions to the individual parts. There is also an option for placing the solutions immediately after the statement of the problem. This may be useful for an example environment, where we want the solution to the example to follow the statement, rather than being hypertext-linked to the solution. Using the forpaper option, we can also make a paper version of the exercises. This method of presenting exercises (and examples) allows the student to attempt the problem before seeing the solution, gives a cleaner presentation of the topic, a presentation not cluttered with solutions to examples and exercises that take away from the main stream of thought. Exercises of this type can be part of a tutorial, or the teacher could simply publish a homework set on the Web (at first, without solutions included, later, with solutions) for the students. The shortquiz environment is used for creating interactive quizzes with *immediate feedback*. As soon as the user enters an answer, that answer is immediately evaluated, the results of the evaluation are communicated to the user. On the other hand, the quiz environment provides macros for creating quizzes with *delayed feedback*, in which the answers are not evaluated until the student has finished the quiz. Both of these two environments enable to optionally include answers and solutions to the questions.

Two general types of questions can be posed by the AcroTEX system: *multiple choice* questions and *objective style* (*fill-in-the-blank*) questions. The multiple choice questions can appear in two basic styles: *link-style* and *form-style*. The link-style uses links to record the choices to the alternatives. This method takes up less space in the PDF file than does the form-style, but the student cannot see the choices made. From that reason, this method is perhaps adequate only for two or three quick questions. For a longer quiz format, one would like to use the form-style – a question with a "checkbox" format, which can be obtained using the *-form of the quiz environment.

There are certain kinds of questions in mathematics and related fields, that the teacher would like more than a multiple choice guess from the student. For example, teachers would occasionally like to ask questions that would require the student to fill in the answer, whether it be numerical or symbolic. Such questions, called open ended or objective questions, increase the level of difficulty for the student. No multiple guessing for this kind of question. The exerquiz package distinguishes between two types of objective questions: a *text fill-in* question that requires to enter a word or phrase as the answer, and a *math fill-in* question that requires a mathematical expression as the answer. If a text fill-in question is posed, the underlying JavaScript compares the user's response against acceptable alternatives, as supplied by the author of the question. If there is a match, the response is deemed correct. When the user's answer and the author's answer are compared, four filtering methods can be used: (1) by default, the author's and user's answers are not filtered in any way (spaces, case, and punctuation are preserved), (2) the author's and user's answers are converted to lower case, any white space and non-word characters are removed, (3) the author's and user's answers are converted to lower case, any white space is removed, (4) the author's and user's answers are stripped of any white space. Moreover, two compare method can be used: (1) by default, the author's and user's answers are compared for an exact match (these answers are filtered before they are compared), (2) the user's response is searched in an attempt to get a substring match with the author's alternatives (additional comparison methods may be added).

A math fill-in question can be posed that requires an answer that is a function of one or more declared variables. The algorithm used for determining the correctness of the answer entered by the user is quite simple: The user's answer and the correct answer are evaluated (as functions) at several randomly selected points in the domain of the correct answer, and then compared. If any of the comparisons differ by more than a preselected amount (an ϵ value), the user's answer is declared incorrect. Otherwise, it is considered correct. When poses a question, the teacher has to determine a correct answer, which must be a numerical value or a function, the number of samples points to be used, (usually 3 or 4 is sufficient), precision required (the ϵ value), and the interval from which to draw the sample points. For example, for the question "Determine $\frac{d}{dx}sin^2(x)$", the code is

```
\begin{oQuestion}{sin}
Determine $\dfrac d{dx} \sin^2(x) =
\RespBoxMath{2*sin(x)*cos(x)}{4}{.0001}{[0,1]}$
\end{oQuestion}
```

Therefore, the correct answer written in valid JavaScript syntax is `2*sin(x)*cos(x)` and evaluation of the user's answer is done by randomly selecting 4 points from the interval $[0, 1]$. If the evaluation at any of the 4 points differs from the evaluation of the correct answer at the same point by more than $\epsilon = 0.0001$, the user's answer is considered wrong. In that manner, `sin(2*x)` is a valid response, too. Before the evaluation, the user's response has to be checked for syntax errors. The user must be careful to enter his/her answer using the correct syntax and the

quiz author has to supply the user with all the necessary instructions how to enter the answer in the correct syntax.

There are several enhancements to the multiple choice and fill-in questions. For fill-in questions, if the document author so wishes, answers can be provided using an "Ans" (Answer) button. For a shortquiz, the "Ans" button is visible anytime and can be clicked at anytime. In the case of a quiz, it is hidden, and after a quiz has been completed, the hidden "Ans" buttons appear. Click on the button will get an answer to the problem. In addition to a correct answer, the quiz author can also include a complete solution to the question. If the "Ans" button has a green border, that means that a question has a solution, and shift-click on the button causes the viewer to jump to the solution. Using the "Ans" button, quizzes with immediate feedback can be created to practice entering the responses using the correct syntax. In this case, a counter that keeps track of incorrect answers is also provided. Quizzes created by the quiz environment can be corrected using the "Correct" (Correction) button. JavaScript is used to correct the quiz. After the quiz is completed and the "Correct" button is pressed, the corrections appear. In the case of a multiple choice question, the correct answer has a green filled circle or a green check. This circle is now outlined by a green rectangle to indicate that this is a link to the solution. Click on the green dot will perform the jump to the solution. The "Correct" button will not work until the user has clicked on "End Quiz". The user can re-take the quiz simply by clicking on "Begin Quiz", the form fields and JavaScript variables will be cleared. A quiz can be protected by the `\NoPeeking` command. If this command is executed in the preamble of the document, or prior to a quiz, then any quiz question with solution will be protected somewhat from prying eyes. In this case, an open page action is placed on the first page of each solution. If the user tries to view a quiz solution before doing the quiz, the Acrobat Reader will automatically change the page to the page containing the quiz and place an alert box on the screen saying that viewing the solution before taking the quiz is not permitted.

In the AcroTEX system, document-level JavaScript is also used to score and grade the student responses. Evaluation of the quizzes created by the exerquiz package is originally done on the client-side, within the web browser or Acrobat Reader. This makes the document entirely self-contained, there is no CGI scripting involved. One of the great advantages here is that the document can be downloaded, brought into the Acrobat Reader and reviewed off-line (the performance of the document is much better within the Acrobat Reader than within a web browser). This kind of quiz is ideal for a do-it-yourself tutorial system, read by a well-motivated student who has the discipline to read the material and to take the quizzes in the spirit in which they are given. However, some educators may wish to use the quizzes created by the exerquiz package for classroom credit. It is necessary, therefore, for the student to be able to submit quiz results to a web server which, in turn, should store the results to a database. This can be attained by means of an additional package eq2db provided by D. P. Story, which redefines the "End Quiz" link or button appropriately so that when the user clicks on it, the results will be sent to a server-side script, which will save the quiz data to a database of some type or send by e-mail to the instructor. The package consists of a LaTeX macro package and ASP scripts eqRecord.asp, for saving the quiz data to a database, and eqEmail.asp, for sending the quiz results to the instructor by e-mail.

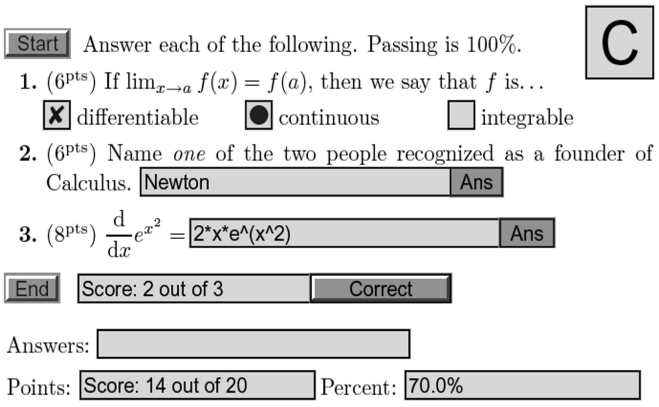

Fig 2. An example of a quiz created by exerquiz package

Another system that produces quizzes in PDF format is MacQTEX, developed by R. Moore and F. Griffin [10]. It is based on the exerquiz package, but there are some interesting innovations. Access to the quizzes is through a web page linked to a CGI script, which verifies the identity of a user and serves a quiz. Each quiz is different – the parameters which determine any numbers used in the questions are randomly generated using the Mathematica software, so that a student may attempt the "same" quiz many times, but each time it will be slightly different. When a quiz is downloaded from a web-site, student is allowed to read and work with the document, using the Acrobat Reader plug-in to his/her favorite web-browser. After starting with the "Begin Quiz" button, answers may be selected and changed, and MacQTEX does not need to communicate with the server until the quiz is completed, as all the interactivity, answer checking and score evaluation takes place inside the PDF quiz document itself. Upon pressing the "End Quiz" button, results of the student's attempts are submitted to a server for recording, provided that the network connection is still available. The document-level JavaScript then provides a means for the student to see which were the correct answers for each question. The staff interface contains a suite of tools which allows lecturers to monitor student progress and to create new quizzes.

More information about all the topics mentioned in this paper can be found in the bibliography listed below.

References

[1] Adobe Systems, Inc., *PostScript Language Reference*, Third Edition, Addison-Wesley Publishing Company, 1999.

[2] P. W. Daly, *Graphics and Colour with LaTeX* (an extract from [7]), Max Planck Institute for Aeronomy, Katlenburg-Lindau, 1998, `http://www.linmpi.mpg.de/~daly/latex/grf.pdf`

[3] M. Goossens, F. Mittelbach and A. Samarin, *The LaTeX Companion*, Addison-Wesley Publishing Company, 1994.

[4] M. Goossens, S. Rahtz, E. M. Gurari, R. Moore and R. S. Sutor, *The LaTeX Web Companion: Integrating TeX, HTML and XML*, Addison-Wesley Publishing Company, 1999.

[5] M. Goossens, S. Rahtz and F. Mittelbach, *The LaTeX Graphics Companion: Illustrating Documents with TeX and Postscript*, Addison-Wesley Publishing Company, 1997.

[6] D. E. Knuth. *Computers & Typesetting* , Millennium Boxed Set, Volumes A to E, Addison-Wesley Publishing Company, 2000.

[7] H. Kopka and P. W. Daly, *Guide to LaTeX*, Third Edition, Addison-Wesley Publishing Company, 1999.

[8] L. Lamport, *LaTeX: A Document Preparation System*, Second Edition, Addison-Wesley Publishing Company, 1994.

[9] T. Merz, *Web Publishing with Acrobat/PDF*, Springer, New York, 1998.

[10] R. Moore and F. Griffin, *MacQTeX: Self-testing quizes, using PDF*, MacQTeX Web Site, 2001, `http://rutherglen.ics.mq.edu.au/~macqtex`

[11] D. P. Story, *Using LaTeX to Create Quality PDF Documents for the World Wide Web*, AcroTeX Web Site, 1998, `http://www.math.uakron.edu/~dpstory/acrotex.html`

[12] D. P. Story, *The AcroTeX eDucation Bundle*, AcroTeX Web Site, 2003, `http://www.math.uakron.edu/~dpstory/acrotex.html`

[13] Hàn Thê Thành, *The pdfTeX Program*, Cahiers GUTenberg 28-29 (1998), 197–210.

[14] M. Wiedmann, *Screen Presentation Tools: Tools for Creating Screen or Online Presentations*, Michael Wiedmann's Web Site, 2003, `http://www.miwie.org/presentations/`

[15] T. van Zandt, *PSTricks – PostScript macros for generic TeX*, User's Guide, 1993, `http://www.tug.org/applications/PSTricks`