serendiPDF with Searchable Math-fields in PDF Documents

Ross Moore
Mathematics Department, Macquarie University, Sydney
ross@maths.mq.edu.au
http://www.maths.mq.edu.au/ ross/

Abstract

serendiPDF is an attempt to make it easier to find the correct way to express complicated mathematics, especially aligned environments, using LaTeX. This is achieved by storing a copy of the LaTeX source for a mathematical environment inside the generated PDF output, in a way that allows it to be easily accessed and copied into the source for other documents. In this way, the full power of "serendipity", as a means for appreciating and learning unfamiliar techniques, becomes available for authors of mathematical LaTeX documents.

The existence of extra (initially hidden) mathematical fields within PDF documents, allows for a solution of the perennial problem of how to search for pieces of mathematics within typeset documents. A solution is presented whereby symbol names, such as `\alpha` (α), `\Gamma` (Γ) and `\Sigma` (Σ), can be located within the extra math fields. The interface behaves just as one would expect from a search-engine, finding fields either anywhere within the document, or limiting the search to just the currently visible page.

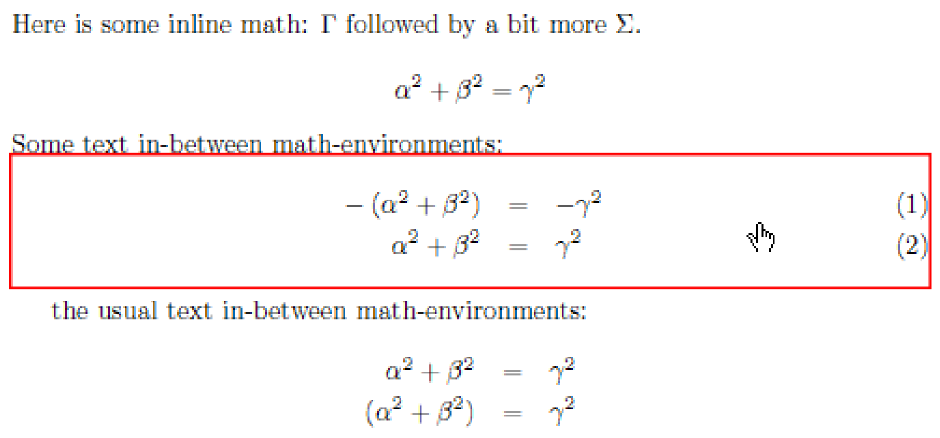
Serendipity

Dictionaries define serendipity as the act of "accidental discovery", such as finding something of value by accident, when actually looking for something else. In the context of modern computing software, with extensive menus and an intricate graphical interface, serendipity clearly plays a rôle in learning how to use the program. When searching through the menus for the way to perform a particular kind of task, one frequently tries out unfamiliar options. In doing this one may not find what was being looked-for, but instead discover how other tasks can be performed. Typically 'features' discovered in this way are remembered, or appreciated, much better than if a manual had been consulted.

With what used to be called WYSIWYG word-processing software ("What You See Is What You Get"), serendipity can play a significant rôle in constructing complex documents. Rather than learning from a manual how to (for example) create a tabular layout or complicated mathematical expression, one just copies something that looks like it does part of what seems to be needed, then makes alterations until it is presenting what is desired. This may not lead to the best possible appearance, or the most efficient (in some sense) coding, but it can get the job done.

To TeX purists this can be anathema—data should be presented in a consistent logical manner, with appropriate mark-up to indicate its meaning, not just appearance. While this is true, it also leads to the perception that TeX (or LaTeX) is difficult, both to use effectively, and to learn—despite its obviously superior output quality. This author contends that the problem is largely due to the 3-step edit–compile–view cycle that is at the core of document preparation using TeX. An inexperienced user can see superb TeX-produced output, but may not know what kind of input source was required to produce it. For all but simple text and paragraphing, it is generally not possible to take the output and reuse it (with appropriate edits) in a new document, without having access to the author's original source coding, or similar work. At least with PDF as the output format, it is possible to capture text using the 'Text Capture' tool. But try to use this tool for mathematics or tables—it just does not help at all.

Such an edit–compile–run cycle used to be the predominant computing paradigm, so it was no surprise that TeX was constructed to work in this way. Nowadays however, many people have used computers effectively for a large number of years without ever (knowingly) having compiled a program; the concept is something completely foreign to them. This can be true of students and academics in all



Here is some inline math: Γ followed by a bit more Σ.

$$\alpha^2+\beta^2=\gamma^2$$

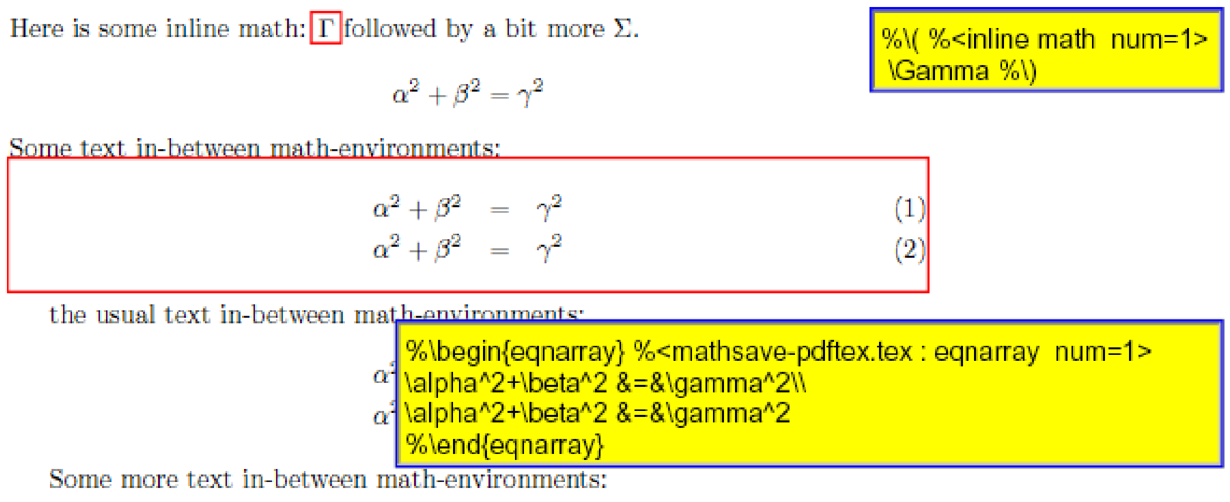
Some text in-between math-environments:

$$\begin{aligned} -(\alpha^2+\beta^2) &= -\gamma^2 && (1)\\ \alpha^2+\beta^2 &= \gamma^2 && (2)\end{aligned}$$

the usual text in-between math-environments:

$$\begin{aligned} \alpha^2+\beta^2 &= \gamma^2\\ (\alpha^2+\beta^2) &= \gamma^2\end{aligned}$$

Figure 1: Moving the mouse over a piece of typeset mathematics causes the outline of an invisible button to be displayed. Clicking on this button toggles the visibility of a field with LaTeX source, see figure 2.



Here is some inline math: Γ followed by a bit more Σ.

```
%\( %<inline math  num=1>
\Gamma %\)
```

$$\alpha^2+\beta^2=\gamma^2$$

Some text in-between math-environments:

$$\begin{aligned} \alpha^2+\beta^2 &= \gamma^2 && (1)\\ \alpha^2+\beta^2 &= \gamma^2 && (2)\end{aligned}$$

the usual text in-between math-environments:

```
%\begin{eqnarray} %<mathsave-pdftex.tex : eqnarray  num=1>
\alpha^2+\beta^2 &=&\gamma^2\\
\alpha^2+\beta^2 &=&\gamma^2
%\end{eqnarray}
```

Some more text in-between math-environments:

Figure 2: In response to a click over a piece of typeset mathematics the visibility of a math-field is toggled. Here we see how the field contains the LaTeX coding for the typeset mathematics, as a complete environment.

fields, so it is not hard to see why TeX has been described as "arcane", and does not occupy the prominent place in publishing that befits the quality of its output. It is the *lack of serendipity* that makes learning TeX seem to be so much harder than for other modern software packages.

The main purpose of serendiPDF is to implement an idea that may help to change this. Now mathematics *can* be recovered from specially prepared PDF documents, using nothing more than the Acrobat Reader [2] provided (free of charge) on all platforms by Adobe Systems Inc. The idea is to include the LaTeX source for mathematics environments as hidden fields within the PDF document. Visual clues indicate the presence of these fields, as indicated in figure 1. In response to a single mouse-click a field can be shown, thereby revealing the LaTeX coding which has then been 'discovered' serendipitously; see figure 2.

A LaTeX document can be prepared such that *all* mathematics coding is included also within these hidden fields. Such a document becomes not only a valuable source of scholarly information on the topic being presented, but also a useful example for learning how to create the high-quality appearance for

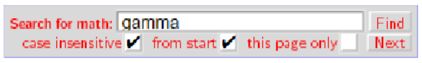


Figure 3: The 'Search for math' control box allows TEX source to be located within math fields. Searches normally look forward from the current page, but check-boxes allow it to cover the whole document or restrict to the one page.

mathematical information of a similar kind. Inexperienced authors can learn from the older masters, not just the intricacies of the meaning embodied in the mathematics, but also how best to present it.

Having the LATEX coding for mathematics available leads to further useful possibilities. For example, it now becomes a simple matter to search for mathematical expressions within a PDF document (see figures 3 and 4); something which was hitherto quite impossible. It should even become possible to develop plug-in software that allows mathematical expressions to be edited in-place.

The serendip package, with insdljs.sty and hyperref.sty

Electronic fill-in forms are have now become quite common, in both HTML and PDF. For the most-part, these rely upon a JavaScript interpreter being available within the web-browser or PDF reader software. (JavaScript is a programming language that handles the appearance of buttons and the showing/hiding of fields and annotations, as well as calculations, and a myriad of other kinds of computing task related to a document and its content.)

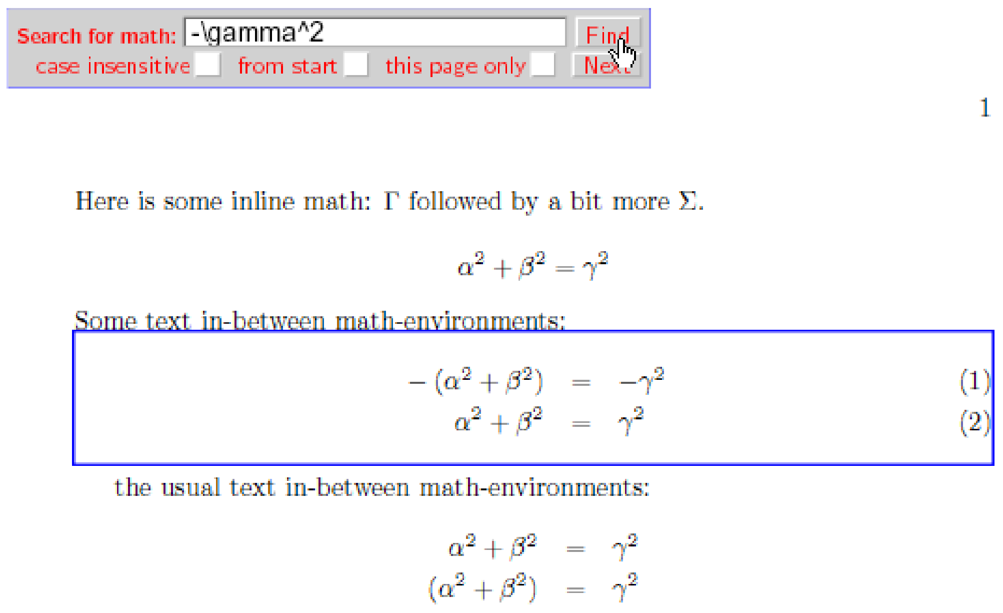
Acrobat Reader [2] has had JavaScript support since version 4.0 [1], and is even more sophisticated in versions 5.0 and later. Donald Story [7] has been pioneering the use of JavaScript within PDF documents generated from TEX source, for several years. His insdljs package (acronym for 'Insert Document-Level JavaScript') provides coding that allows functions and procedures to be included within PDF documents, using any of dvips (+ Ghostscript or Distiller) or dvipdfm or pdfTEX as the engine producing the final PDF output. In fact with pdfTEX [4] the inclusion of JavaScript is relatively straight-forward, using just the hyperref package to help specify fields and buttons. With other drivers, the pdfmark [3] technique is used extensively.

For placing mathematics code into fields, the serendip package builds upon the support for fields, buttons and JavaScript in both the insdljs and the hyperref packages. The serendip package works with LATEX math environments, such as `\(....\)` for inline, and `\[....\]` for displayed math, as well as the equation, eqnarray and displaymath environments. It also works with the outer-level environments and alignments defined within the amsmath package. It does this by redefining, in a non-destructive way, the behaviour of some LATEX and AMS macros.

For example, with the equation environment, as started by `\begin{equation}`, it is the macro `\equation` which is redefined to behave as follows.

1. Open a new level of grouping and redefine the `\catcode`s of non-alphabetic characters.
2. Read to the corresponding `\end{equation}`, to get the complete source for this environment.
3. Store this LATEX source as a list of tokens, to be later written as plain text into a hidden field within the PDF document being constructed.
4. Estimate the size (both height and width) required for the text field by typesetting the tokens in a `\hbox` using a fixed-width font then measuring the result.
5. Write the list of tokens into a file (with `.mth` extension) so it can be re-read by TEX with the usual `\catcode` values.
6. Construct the text-field containing the LATEX source, named sequentially with all equation environments. and positioned using `\marginpar`.
7. Close the grouping level, reverting `\catcode`s to their normal TEX values.
8. Start a new inner `\vbox`, to hold the typeset mathematics.
9. Read-in the contents of the `.mth` file for typesetting, using a stored pointer to the usual expansion of `\equation`.
10. Measure the size of the resulting `\vbox` and construct a button of this same size.
11. Place the `\vbox` onto the page with the correct amounts of preceding and trailing glue.
12. Remove trailing glue, remembering how much was used.
13. Place the button also onto the page, directly over the typeset mathematics. This button will be used in the final PDF to toggle visibility of the text-field.
14. Replace the trailing glue, so that the environment interacts correctly with material following afterwards.

The resulting page may differ slightly in the stretchability of the glue around the environment. Mostly this is not noticeable at all. Other environments are handled similarly, except for inline mathematics, where the typesetting is done within an `\hbox`, which is measured and later placed onto the page. Thus inline-math environments must occur entirely



Here is some inline math: Γ followed by a bit more Σ.

$$\alpha^2 + \beta^2 = \gamma^2$$

Some text in-between math-environments:

$$\begin{aligned} -(\alpha^2 + \beta^2) &= -\gamma^2 \qquad (1)\\ \alpha^2 + \beta^2 &= \gamma^2 \qquad (2)\end{aligned}$$

the usual text in-between math-environments:

$$\begin{aligned} \alpha^2 + \beta^2 &= \gamma^2\\ (\alpha^2 + \beta^2) &= \gamma^2\end{aligned}$$

Figure 4: When the specified math coding has been found, the environment is indicated by highlighting the border of its overlying button, and giving focus to that button. The field itself is not shown until the button is pressed. Repeated use of 'Next', instead of 'Find', cycles through subsequent occurrences.

within a single line; any breaks need to be forced, resulting in two fields and a button for each.

Searchable Mathematics

One consequence of having the LaTeX code for mathematics available in text-fields within the PDF document, is that now it becomes possible to search these fields using JavaScript procedures. The mathsrch package constructs a console, as shown in figure 3. This allows mathematical expressions to be found by searching forward within the math-fields embedded within the current PDF document.

By checking a box, a search can be modified to restrict to just the current page, or to cover the whole document, starting from the beginning. Using the 'Next' button, rather than 'Find', allows all occurrences of a particular expression can be located sequentially. To indicate a found expression, the mathematical environment is indicated by outlining its button, as shown in figure 4. If the LaTeX code itself needs to be shown, then an extra click is required on this outlined button.

Placing the search-console is not an automatic consequence of loading the mathsrch package. Certainly this constructs the console as the contents of a TeX box register, called `\MathSearchBox`. This box can be placed anywhere on a LaTeX page, using the command `\MathSearch`, which is just a macro that expands to `\copy\MathSearchBox`. (It is important to use `\copy`, rather than `\box`, so that that console can be used repeatedly on different pages.)

Since space characters are usually ignored in (LA)TeX math-mode source, some flexibility is built into the searching mechanism. A space token in the search-string is not required to match in the math-fields; in fact, it can match any number of spaces, including none at all. For example, `x+y` matches only `x+y` in a math-field; but `x + y` will find any of `x+y`, `x +y`, `x+ y`, `x + y`, as well as `x  +y` and `x  +  y`, and other strings having more spaces.

The mathsrch package also defines a command `\MathSearchInHeader`, which can put a console on every page, situated neatly above the header and abutting into the left-hand margin. More precisely, `\MathSearchInHeader` calls upon another macro, `\PlaceMathSearchBoxInHeader`, which expands as:

```
\newcommand{\PlaceMathSearchBoxInHeader}{%
  \pagestyle{myheadings}%
  \markboth{\protect\MathSearch\hfill}%
   {\protect\MathSearch\hfill}}
```

From this it can be seen that the `\pagestyle` is set to be myheadings, and the header contents are given explicitly. If other page-styles are being used, then it is appropriate to make a re-definition:

```
\renewcommand{\PlaceMathSearchBoxInHeader}{%
  \pagestyle{.....}%
  \markboth{.....}{.....}}
```

to accommodate the desired page-style and header.

Customisation of the search-panel is also possible; e.g., for a language other than English, or to change width and colours. A file msearch.cfg is read, if it can be found on any of the usual LaTeX search-paths. The customisable para meters are provided with default values as follows:

```
\providecommand{\msearchString}{Search for math:}
\providecommand{\msearchFindString}{ Find }
\providecommand{\msearchNextString}{ Next }
\providecommand{\msearchPageString}{this page only}
\providecommand{\msearchCaseString}{case insensitive}
\providecommand{\msearchStartString}{from start}
\providecommand{\msearchText}{put TeX code here}
\providecommand{\msearchWidth}{3.25in}
\providecommand{\msearchFGcolor}{red}
\definecolor{ltgray}{gray}{.85}
\providecommand{\msearchBGcolor}{ltgray}
\providecommand{\msearchBorderColor}{blue}
\providecommand{\msearchBorderOpts}{borderstyle=B,
  borderwidth=1, bordercolor= .85 .85 .85}
```

Any of the above macros can be given different expansions either within mathsrch.cfg, or *before* the mathsrch package is loaded, or values changed *after* the package has been read, using `\renewcommand`.

Future Developments

The original intention for having math-fields is to allow the less-obvious parts of a document's TeX source to be distributed along with the final PDF. This is meant primarily as a teaching aid. It remains to be seen whether it will indeed be used in this way; or if other applications are found, once this extra enrichment of PDF documents becomes more widespread.

For example, the code from a math-field can be edited in-place, and subsequently used to generate a modified form of the mathematical environment. This can be done in an external program or utility, such as the 'Equation Service' by Bob Rowlands [6] for Macintosh OS X, that works as a process callable from other running applications. (In effect, pdfTeX generates a new image of just the modified mathematics.) It should be possible to use the plug-in technology for Adobe's Acrobat (full version, not just the Reader), to include such an image into the original PDF, for display in the same location as the typeset mathematics from which it was derived. By merging this image as an update to the original document, we would have what is effectively PDF editing capabilities for touching-up TeX-typeset mathematics.

Acknowledgements The author wishes to thank Donald Story, for many email discussions and help with suggestions about how to implement various technical ideas using JavaScript. Without his work on insdljs and the exerquiz packages, and testing of example documents, the serendip and mathsrch packages described here would have taken much longer to complete.

References

[1] Adobe Systems Inc.; "Acrobat Forms JavaScript Object Specification, Version 4.0"; Technical Note #5186; Revised: January 27, 1999.

[2] Adobe Systems Inc.; Acrobat Reader, viewer for PDF format documents, available free of charge from http://www.adobe.com/.

[3] Adobe Systems Inc.; "`pdfmark` Reference Manual"; Technical Note #5150; Adobe Developer Relations; Revised: March 4, 1999.

[4] Hàn, Thế Thành; pdfTeX, free software for generating documents in PDF format, based on the TeX typesetting system. Available for all computing platforms; see http://www.tug.org/applications/pdftex/.

[5] Netscape Communications Corporation; Netcape JavaScript Reference, 1997; online at http://developer.netscape.com/docs/manuals/communicator/jsref/toc.htm.

[6] Rowland, Bob; 'Equation Service', program for Macintosh OS X to produce small PDF images of TeX-typeset mathematics or text; version 0.5b, 2002. Software available online from http://www.esm.psu.edu/mac-tex/EquationService/.

[7] Story, Donald; `exerquiz` & AcroTeX, packages for including special effects in PDF documents, using TeX and LaTeX. Dept. of Mathematics and Computer Science, University of Akron. Software available online from http://www.math.uakron.edu/~dpstory/webeq.html.

[8] Story, Donald; "Techniques of Introducing Document-level JavaScript into a PDF file from a LaTeX Source," *Tugboat,* 22(3) pp. 161-167, Proceedings of TUG 2001, TeX Users Group Annual Meeting, Delaware, August 2001.