DBTeXMath

Allin Cottrell
Wake Forest University
Department of Economics
November 19, 2001

Table of Contents

Executive Summary

If you write SGML documents using the DocBook DTD, the files offered here let you embed TeX equations directly in your SGML source files, and arrange for the mathematical notation to be fed directly to TeX on output — hence avoiding both (a) the need to code mathematics in MathML on the input side, and (b) the need to rely upon experimental and unfinished dsssl-based mathematical typesetting code. Provision is made for substituting graphical variants of mathematical formulae in the case of output to HTML.

Rationale

SGML, using the DocBook DTD, provides an excellent canonical format for storing complex documents (e.g. the manual for the econometrics program `gretl`, which I maintain), especially if one wants to be able to produce both printable (e.g. PDF) and web-viewable (HTML) versions of these documents on demand.

If these documents contain a significant amount of mathematical notation, however, one is likely to run into problems. There is a means of dealing with mathematics in the context of (semi-)standard DocBook — namely, using MathML mark-up — but this approach has two big problems.

§ MathML mark-up is *extremely* verbose. It is impossible to write and edit MathML by hand, other than as an "exercise," or for very simple mathematical expressions.

§ Even if you can get your math into MathML notation, somehow or other, you then face serious problems in getting it properly typeset in the printable output.

I will expand on each of these problems in turn.

Producing MathML: The inordinate difficulty of writing in MathML directly means that for even moderately complex expressions one is forced to use a GUI equation editor of some sort. But this in turn has two drawbacks, from my point of view. First, in the realm of free, open-source software, such editors are hard to come by. The experimental `Amaya` is, to my knowledge, the only such open-source tool available. Second, even if a suitable equation editor is available, writing mathematics in this way does not sit well with my preferred

mode of document preparation, namely WYSIWYW or "What You See Is What You Wrote," for example editing the SGML source in `emacs` with the help of a suitable mode.

Typesetting from MathML: A dsssl engine such as `openjade` can turn the MathML into TeX for you but the results are likely to be disappointing, particularly if you are used to typesetting mathematics using TeX itself. TeX's native mathematical typesetting is near-perfect, only occasionally requiring manual tweaking to achieve optimal results; it is also rather comprehensive, with the aid of the `AMS` (American Mathematical Society) extensions if need be. But if you take the route of MathML to TeX via dsssl and `jade`, the specifics of the math typesetting must be handled by the dsssl stylesheet. David Carlisle put some work into this a few years back (for which we can be grateful), but he didn't finish the job and nobody else has done so since. Thus if you send MathML through `jade` to TeX you are likely to find (a) that those elements that are recognized by the stylesheets are typeset less adeptly than by TeX itself (with clumsy-looking spacing), while (b) various important elements may not be recognized at all. For example in my field of statistics the overbar (denoting the arithmetic mean) is a common modifier, but it is simply ignored. Other formulations common in statistics are also ignored, or are not dealt with properly, so this route is not really usable for me.

Implementation

"DBTeXMath" is implemented via (a) a minor hack to the DocBook dsssl stylesheets as offered by Norman Walsh, and (b) a couple of `perl` scripts, one to "unescape" TeX math passed through `jade` and one to generate graphical versions of TeX math for use in making HTML.[1]

The idea is that TeX math is written into the <`alt`> element within equations. The stylesheet hacks produce the effect that (a) the literal TeX math gets passed through when producing PDF or postscript, while (b) it gets ignored (in favor of a PNG rendition of the math) when producing HTML.

Using `perl` to post-process the output of a dsssl engine is truly an ugly hack, but I think I can rationalize it! While a "cleaner" solution would be to modify `openjade` (adding an appropriate switch) there are a couple of things against this: one, I'm no `C++` jockey and I simply don't have time to learn how to hack on `openjade` effectively; and two, by offering "DBTeXMath" as a small package of scripts rather than as a patch to `openjade` of questionable acceptability I'm (hopefully) making it easier for others to try it out and see if it works for them.

The files

The package consists of the following files:

`TeXMath.dsl`

dsssl stylesheet fragment for use in producing TeX output (for further processing by `jadetex` in order to make PDF or postscript).

`HTMLMath.dsl`

dsssl stylesheet fragment for use in producing HTML output. There are some configurable options in here, governing the appearance of the equation bitmaps.

1. I'm grateful to Jirka Kosek for pointing out that an additional element included in the original version of this package — namely a modification to the DocBook DTD — was not in fact necessary.

`sample-both.dsl`

skeleton dsssl stylesheet that calls `TeXMath.dsl` or `HTMLMath.dsl` depending on whether TeX or HTML output is required. I have kept this very simple; you can add your own stylistic customizations, or use it as a model for including the dsssl math fragments in your own stylesheet.

`sampler.sgml`

sample SGML source file showing the use of the `alt` element for math.

`unescape_math.pl`

`perl` script to be run after running `jade` but before running `jadetex`.

`texmath2png.pl`

`perl` script to generate on the fly PNG images of equations in the SGML source file. Requires `latex`, `dvips` and the ImageMagick `convert` program.

`jadetex.cfg`

sample configuration file for `pdfjadetex`.

`Makefile`

used to compile the sample document; shows the intended order of events and dependencies when using this approach.

`about.pdf`

this explanatory document in PDF format.

`dbtexmath-0.2.tar.gz`

gzipped archive containing current versions of all of the above files.

Usage

To try out the package you should first explode the `tar.gz` file in a suitable location (it will create a subdirectory named `dbtexmath-0.2`). Then check the Makefile for compatibility with your system, and check the local paths to the DocBook stylesheets in `sample-both.dsl`. You may then compile the sample document with `make pdf` (PDF output) or `make html` (HTML output to the subdirectory `html_out`). To compile your own SGML document using this system, copy the Makefile and edit it appropriately, copy your document into the `dbtexmath` directory, and compile it.

Samples

Here is a little bit of inline math: $C = \alpha + \beta Y^{\gamma} + \epsilon$. It is embedded in the text. The SGML source looks like this:

```
<para>Here is a little bit of inline math:
  <inlineequation>
    <alt>$C = \alpha + \beta Y^{\gamma} + \epsilon$</alt>
    <graphic align="center" fileref="figures/cfunc.png"/>
  </inlineequation>.
  It is embedded in the text.
</para>
```

The `align="center"` attribute here has the effect of centering the graphic vertically; without this an equation is likely to sit above the baseline in the HTML output. Getting a graphic centered horizontally on the HTML "page" (not relevant for inline math, but it would be nice for displayed equations) is more difficult. If anyone knows how to do this, please let me know!

Now here is a bit of displayed math

$$F_{2,T-k} = \frac{(ESS_r - ESS_u)/2}{ESS_u/(T-k)}$$

Note that for this sort of display you get excessive vertical spacing if you bracket the math with double dollar signs, `$$`. The spacing is better if you use `\[` and `\]` instead. The source is:

```
<para>Now here is a bit of displayed math
  <informalequation>
    <alt>
      \[F_{2,T-k}=\frac{(ESS_r-ESS_u)/2}{ESS_u/(T-k)}\]
    </alt>
    <graphic fileref="figures/fstat.png"/>
  </informalequation>
Note that for this sort of display you get excessive...
```

Now here is a slightly more complex piece of displayed math, for which we adjust the spacing using

```
\renewcommand{\arraystretch}{1.3}
\setlength{\arraycolsep}{.05in}
```

within the `alt` element:

$$\begin{bmatrix} \dot{L} \\ \dot{C} \end{bmatrix} = \frac{1}{-\alpha} \begin{bmatrix} af & bfh \\ 1 & b(fh-k) \end{bmatrix} \begin{bmatrix} L \\ C \end{bmatrix} - \frac{1}{\alpha} \begin{bmatrix} afh & 0 \\ h & \alpha \end{bmatrix} \begin{bmatrix} \dot{m} \\ r_w \end{bmatrix}$$

Finally we try using the formal `<equation>` tag with a title:

Equation 1. Test equation

$$\hat{\beta} = \frac{\sum_{t=1}^{n} x_t y_t - \bar{y} \sum_{t=1}^{n} x_t}{\sum_{t=1}^{n} x_t^2 - \bar{x} \sum_{t=1}^{n} x_t}$$

In all cases the `graphic` is ignored if you're using `jade`'s TeX back-end, while the `alt` is ignored and the graphic used when producing HTML. This document (as you might have guessed) was produced using this system.

Technical details

Just in case you're interested — or want to try to improve the system.

The TeX math dsssl fragment contains the following instructions for processing the `equation` element (and similar instructions for `informalequation` and `inlineequation`).

```
(element (equation graphic) (empty-sosofo))
(element (equation alt)
```

```
 (make display-group
   (literal "BEGINTEXMATH")
   (literal (data (current-node)))
   (literal "ENDTEXMATH")))
```

The `(literal (data (current-node)))` command asks `jade` to dump the math within the `alt` element directly into the `tex` output file. By itself, however, this does not do what we want: `jade` "escapes" the TeX string (for instance, turning a backslash into `\char92{}` and a caret into `\char94{}`), so that unless we take special measures we will end up with a typeset version of the *source* for the equation rather than the typeset equation. The script `unescape_math.pl` is used to undo this escaping; the markers `BEGINTEXMATH` and `ENDTEXMATH` are written into the output file so that the `perl` script can easily locate the strings to be treated in this way.

The HTML math dsssl fragment arranges for `jade` to make two passes through the SGML document — one to output the HTML itself and a second to output a listing of the equations found in the document:

```
(root
    (make sequence
      (process-children)
      (process-math)))
```

The second pass results in the writing of an auxiliary file, `equation-list.sgml`, containing various parameters governing the conversion of the equations into PNG images along with a listing of the equations, giving for each one the filename to use for the graphic and the TeX math content. This auxiliary file is itself an SGML document, which should validate against the following DTD:

```
<!DOCTYPE equation-set [
<!ELEMENT equation-set - - (texequation+)>
<!ATTLIST equation-set
          latexopt CDATA #IMPLIED
          density CDATA #IMPLIED
          usepackage CDATA #IMPLIED>
<!ELEMENT texequation - - (#PCDATA)>
<!ATTLIST texequation fileref CDATA #REQUIRED>
]>
```

Below is a simple example of `equation-list.sgml` as produced by `jade`. Note that the initial parameters to the `equation-set`, namely `latexopt`, `density` and `usepackage`, are set in `HTMLMath.dsl`. The default values are `"12pt"`, `"96x96"` and the empty string `""` respectively.

```
<equation-set
latexopt="12pt"
density="96x96"
usepackage="mathtime"
><texequation
fileref="figures/fstat.png"
>         \[F_{2,T-k}=\frac{(ESS_r-ESS_u)/2}{ESS_u/(T-k)}\]
        </texequation
></equation-set
>
```

The `perl` script `texmath2png.pl` is used to parse this listing. For each `texequation` found, the script does the following:

—writes out a `tex` file;

—runs `latex` on this file;

—runs `dvips` on the resulting `dvi` file, using the `-E` flag to produce encapsulated postscript;

—runs `convert` (which is part of ImageMagick) to turn the postscript into PNG; and

—cleans up the temporary files generated in the previous steps.