

Interactive Mathematics via the Web using MathML

Francis J. Wright

School of Mathematical Sciences

Queen Mary and Westfield College, University of London

Mile End Road, London E1 4NS, UK.

F.J.Wright@QMW.ac.uk

<http://centaur.maths.qmw.ac.uk/>

Abstract

MathML is a mathematical markup language intended for displaying mathematics in web browsers. At present, it can be used to display mathematics generated dynamically in response to interactive queries only if the browsing and generating facilities are chosen carefully. This paper examines the background and possible options, and describes some of the details of the use of MathML to display the output from a web-based demonstration of an ordinary differential equation solver running in REDUCE. Two screenshots showing examples of rendered MathML output are included.

1 Introduction

It is now in principle quite easy to provide interactive “mathematics servers” on the World Wide Web [1]. The problem can be input via a web-based form, processed by a remote computation server, and the output returned to the user’s web browser. The remaining problems are ones of detail, such as how best to render the mathematical output. The user can reasonably expect an interface that is comparable with current computer algebra systems used directly. Therefore, one can expect the user to type linearized mathematical notation into a form input region, but the user will expect near-typeset quality output.

In a web context, the obvious language to use to convey mathematics is MathML [4], the mathematical extension of HTML [2] to support high-quality mathematical output directly. Other good alternatives would be $\text{T}_\text{E}\text{X}$ [34] or OpenMath [15]. Less good alternatives would be PDF, PostScript, DVI and proprietary formats, such as those used by word-processors. These formats are excellent for printed documents, and range from good to acceptable for static documents to be accessed via the web, but (roughly speaking) the less similar to HTML a format is the poorer it is for dynamic documents. The $\text{L}^{\text{A}}\text{T}_\text{E}\text{X}$ Web Companion [38], published about a year ago, is a very useful background reference for many of the issues discussed in this paper.

The precise context of this paper is a web-based demonstration of an ordinary differential equation (ODE) solver using the computer algebra system (CAS) REDUCE [24], which I developed for the CATHODE project [16]. However, the focus for this paper is on interactive mathematics and output rendering rather than on the capabilities of the solver itself (which are still limited). The demonstration originally output only plain text. Subsequently I added a $\text{T}_\text{E}\text{X}$ output option, to be rendered by the IBM techexplorer plug-in [17],

and recently I have added various MathML output options. The demo can be accessed as

http://centaur.maths.qmw.ac.uk/CATHODE/ODESolve_demo.html

The first step in designing a web-based interactive mathematics system is to consider the browsing capabilities and the facilities for generating mathematical output markup that are either actually available or may soon be available, and I discuss these issues in the next three sections of the paper. I consider only a subset of the freely available applications for rendering MathML; among these, the combination of Netscape Navigator 4.7 [19] and the IBM techexplorer 2.5.2 plug-in [17] seems to work best at present. After that, I discuss the browser interface to MathML and the current status of HTML, XML and XHTML. I then outline how to implement an HTML form interface to a CAS and present a few details of the way that I handle the use of MathML in my web demonstration and the perl CGI program that drives it, and display two screen-shots showing examples of how the MathML output is rendered. I conclude with some thoughts on how further to improve this demo and on the status of using MathML in practice.

Everything to do with the web is changing very quickly; the information in this paper is the best that I could obtain in May 2000, but some of the details may quickly become out of date!

2 $\text{T}_\text{E}\text{X}$ versus OpenMath versus MathML

In this section I will outline very briefly three systems for mathematical markup, in chronological order.

$\text{T}_\text{E}\text{X}$ [34] was designed about 20 years ago by Donald Knuth for high-quality typesetting of mathematical text. It is therefore intended only to describe the presentation or appearance of a text and not its content or semantics. It provides very good automatic formatting, but it also allows the user to over-ride its automatic layout rules. It is intended to be written directly by humans and so is fairly succinct, but it is intended to be processed by machine (it is also a programming language) and so has a regular syntax. Mathematical typography is extremely well integrated into $\text{T}_\text{E}\text{X}$ and it can be used to describe a complete document. It is very familiar to most mathematical scientists. $\text{T}_\text{E}\text{X}$ software has always been Open Source, since long before this was a popular concept, and good quality software is available freely for most platforms. However, a document written in $\text{T}_\text{E}\text{X}$ (such as this paper) is normally processed

in two steps: the first step produces a device independent intermediate binary (DVI) file; the second step renders the DVI file on paper or a display screen. One notable exception is IBM techexplorer, which renders \TeX source directly in a single pass but sacrifices some of the more subtle features of \TeX in the process.

OpenMath [15] is a project initiated in the early 1990's by the Maple developers to facilitate communication of mathematical expressions among algebraic computing systems, mathematical display engines and related software. It focuses primarily on unambiguous transmission of mathematical semantics and regards the presentation of the mathematics as secondary. This preoccupation with avoiding ambiguity tends to make OpenMath verbose. Unfortunately, OpenMath was not originally taken up with the enthusiasm that it deserved and its initial development was slow. This may be a consequence of attempting to communicate among proprietary systems, all of which have proprietary data formats and commercial interests to protect.

Neither \TeX nor OpenMath is primarily oriented toward the web: \TeX predates the web by at least 10 years and OpenMath was initially more concerned with direct interprocess communication using mechanisms such as sockets.¹ Normal \TeX syntax is completely different from the SGML-based syntax that dominates documents intended to be browsed via the web and it seems very unlikely that it will ever be directly supported by any web browser. The long-term prospects for OpenMath are less clear, but MathML looks set to steal a large chunk of its potential applications. MathML is only a few years old and arose out of abortive attempts to include mathematics more directly in HTML.

The principal advantage of MathML is that it was designed precisely for the task in hand, namely inclusion of mathematics in documents intended to be browsed via the web. Hence, it should soon be rendered directly by web browsers, and the inclusion of mathematics in a web document should be as smooth as the inclusion of other objects, such as tables and images. Moreover, there are essentially two variants of MathML: presentation and content (or semantic) markup. They can also be mixed. It should soon be equally easy to generate and render both forms. Presentation markup provides more precise control over the display of the mathematics, in much the way that \TeX does, whereas content markup retains the mathematical significant (i.e. the semantics, although not as reliably as does OpenMath). Hence, content markup allows fairly reliable machine processing of the mathematics, at least in simple contexts, whereas the semantics of presentation markup may be too ambiguous.

It is likely that presentation MathML will soon compete directly with \TeX in terms of display quality and controllability (but not succinctness). Content MathML successfully competes with OpenMath for communication of simple mathematics, although it does not at present have the power to convey unambiguously arbitrary mathematical concepts, whereas in principle OpenMath does. Probably, the use of MathML to convey semantics should be restricted to situations where the context removes any ambiguity. It clearly combines some of the advantages of \TeX with some of the advantages of OpenMath, and MathML and OpenMath could effectively merge at some future date.

¹However, the only OpenMath communication mechanism that appears to work reliably is via files [39].

3 Browser Technology

As far as I am aware, neither of the "mainstream" browsers, namely Netscape Navigator [19] and Microsoft Internet Explorer [18], currently supports MathML, although it is not hard to find pronouncements by MathML exponents that they will both soon support it. Beta test releases of the next versions of both browsers (namely Navigator 6 and Internet Explorer 5.5) are available, but I have not been able to find any reference to MathML in the descriptions of either.

However, Navigator 6 is based on an Open Source project called Mozilla [20], which does aim to support MathML [21]. Test builds that include the MathML support are freely available, and are expected to form the basis for a MathML-enabled version of Navigator at some unspecified future date. At present, Mozilla has a number of limitations. It renders only MathML presentation, not content, markup and it renders MathML only within XML documents (see §6). See [23] for some examples of MathML that is rendered by Mozilla. Whilst Mozilla is able to render my demo page, i.e. the page with the input form, it has proved very difficult to persuade it to display the output even in plain text format. Since the mainstream browsers do not currently accept XML, I do not at present generate XML output.

Amaya [6] is the "reference" web editor and browser provided by the "World Wide Web Consortium" (W3C) [1], which is freely available and supports MathML. Version 3.1 was released on 19 April 2000, but my experience suggests that it is unfortunately not yet robust enough for serious use. It has trouble rendering and following the links in some of the web pages on my CATHODE web site, and the demo page itself seems to crash Amaya 3.1, which is disappointing! (This may be because of my deviations from strict HTML 4.0; nevertheless, these pages render satisfactorily in Navigator and Explorer.) Like Mozilla, it renders only MathML presentation, not content, markup, although unlike Mozilla it renders MathML within XHTML documents (see §6).

Neither of the two MathML-enabled browsers described above seems to be robust enough yet to function as an interactive mathematics client.

An alternative approach is to use a browser plug-in or helper application to render the MathML. A good choice is the IBM techexplorer plug-in, the "introductory" version of which is available free. IBM techexplorer supports a subset of the presentation elements of MathML, as well as the complete collection of content elements. It also renders a large subset of \TeX , \LaTeX and AMS- \LaTeX , and I have used it to render the \TeX output from my demos for some time. My experience suggests that techexplorer is robust when used with Netscape Navigator, making this combination an obvious choice for rendering both \TeX and MathML. There are other plug-ins and applets available for rendering MathML [5], which I will not explore here.

4 Generating Mathematical Output Markup

\TeX can be (and usually is) written by hand, but OpenMath and MathML are not primarily intended to be. However, in the context of interactive mathematics via the web the mathematical markup needs to be generated by computer, regardless of whether it would be feasible to generate it by hand. There are essentially two alternatives: either generate the required markup directly, or generate some other output format and translate it [27] in a post-processing phase. The

latter is in principle easy enough to implement under control of a CGI program given a translation program, but of course would be slower and less elegant than the former option.

\TeX is so well established that most computer algebra systems provide facilities for generating output in \TeX . In the case of REDUCE, the intermediate markup used for typeset-quality interactive display is already \TeX (with some very minor modifications). Hence, it is very easy to generate \TeX output from REDUCE that is suitable for rendering by techexplorer: the standard typeset-quality output requires only minimal post-processing. This is the mechanism used to generate \TeX output for all my REDUCE-based CATH-ODE demos.

A MathML interface package [25] is distributed with REDUCE 3.7 [24], which was released on 30 April 1999. It was developed by Luis Alvarez-Sobreviela and Winfried Neun at the Konrad-Zuse-Zentrum für Informationstechnik in Berlin. This makes it almost as easy in REDUCE to provide MathML output as \TeX output. Mathematica 4 also provides MathML output [26]. The developers of most other computer algebra systems claim that they will support MathML very soon, and some may already do so.

However, the REDUCE MathML package generates only content markup. This is not a problem if techexplorer is used as the renderer, but it may be the case that the mainstream browsers will (at least at first) render only presentation markup. It would therefore be useful to be able to generate presentation markup, which there are two ways to do. The elegant way would be to extend the existing MathML package; the quick and dirty way would be to convert some other output format to presentation MathML using an existing converter. For example, it would probably not be too hard to convert the present \TeX output to presentation MathML. Among potential translation facilities, Omega [28] and TeX4ht [29] both look good and are freely available – see also [38].

Actual support for OpenMath in *currently distributed* computer algebra systems seems to be even more tenuous than that for MathML, although a lot of experimental work has been and is being done [30].

5 The MathML Interface

The next issue to address is how to put MathML markup in a document so that a browser will accept and render it.

The definitive documentation about all things to do with the web is provided by W3C [1], and this web site is an excellent starting point for information about MathML. The current specification of MathML is documented in the “Mathematical Markup Language (MathML) 1.01 Specification” [9] (a draft of version 2.0 is also available). The internal details of MathML are not of primary importance for this paper, but the interface to MathML is, and this is (currently) documented in Chapter 7, “The MathML Interface” [10].

Several things complicate the use of MathML at present. They arise mainly from the fact that web technology is developing extremely rapidly, and MathML in particular is in a transitional phase in which it relies on other technology (XML) that is also still developing.

The principal markup language used on the web is HTML (HyperText Markup Language), and MathML was originally intended to be embedded within HTML. HTML is an instance of SGML (Standard Generalized Markup Language), which was well established before the web came

into being. However, SGML is somewhat cumbersome, and recent development has led to the simpler and more streamlined markup language XML (eXtensible Markup Language). Recently developed markup languages for the web, such as MathML, are therefore instances of XML. HTML and XML are both instances of SGML, but neither is an instance of the other.

MathML specifies a single top-level math element, which encapsulates each instance of MathML markup within a document. Hence, MathML should in principle be included within HTML like this:

```
<math>
  <!-- MathML content here -->
</math>
```

The `<math>` tag can carry various attributes to specify additional markup such as style details, in particular a *mode* attribute that can have as its value either *display* or *inline*. However, web browsers simply ignore markup that they do not understand, and this is what currently happens to MathML markup included within HTML as above: all the markup is ignored and the content is rendered as HTML (i.e. normal text), and so loses all the positioning and much of the special symbolism used in mathematics. The problem then is how to include MathML markup within HTML so that it is recognised as MathML and rendered as mathematics.

There is a page set up as a demonstration of MathML in Amaya [7] that includes Maxwell’s equations as MathML contained in this way within HTML. As might be expected, Amaya 3.1 renders it correctly. (Indeed, Amaya provides WYSIWYG editing of the page, including the math component!) However, neither Netscape Navigator 4.7 nor Microsoft Internet Explorer 5.0 renders the MathML (even with techexplorer installed). Moreover, a MathML-enabled Milestone 15 (completed April 18, 2000) test build of Mozilla/5.0 also failed to rendered the MathML. The reason, as explained in “Implementation of MathML in Mozilla: Progress Report” [22], is that Mozilla currently renders MathML only via XML documents and does not yet support the cohabitation of XML and HTML within the same document.

I am sure that it will not be long before once can use the `<math>` tag alone to include MathML within (X)HTML and have it rendered by any of several browsers, even when generated dynamically via a CGI application. But at the time of writing this seems not to be feasible. At present, it appears to be necessary to use one of the mainstream browsers (Navigator or Explorer) that is robust enough to handle CGI applications, together with a plug-in to render the MathML, and I will consider only the use of techexplorer.

One easy way to use MathML is to return a document consisting only of mathematics and so written entirely in MathML, and give the document a MIME type that can be handled by a browser. In particular, techexplorer can handle “naked MathML” documents with MIME type `text/mathml`, and the examples included in the techexplorer distribution include some such files with extension `.mml` that are registered to correspond to this MIME type. This is the same as the technique that I use to render dynamic \TeX documents. The difference is that \TeX can include non-mathematical text, whereas MathML (essentially) cannot, and most of the difficulty of using MathML is how to include it in a text document. The normal output from the ODESolve demo includes some header information about

the solver, and it can optionally include a transcript showing the actual input to the solver. This would all be lost using only MathML output.

MathML (or $\text{T}_\text{E}\text{X}$) can be embedded in HTML to be rendered by techexplorer by using the HTML `<embed>` tag. The MathML can be contained in a separate file, analogous to the normal use of the `` tag, but this would not be very convenient for dynamically generated CGI output, mainly because of the difficulty of deciding when to delete the temporary MathML file. Moreover, this approach seems to be contrary to the spirit of MathML, which is designed to treat the mathematics as an integral part of the document, and it would require two http transactions to return the output instead of one. However, it may be the only way at present to render dynamic MathML using techexplorer as a plug-in with Internet Explorer.

Alternatively, the MathML can be included inline in the HTML document as follows. This example is taken from the techexplorer documentation:

```
<embed type="text/mathml"
  mmldata="<math><apply><transpose/><matrix>
  <matrixrow>
    <cn>2</cn><apply><root/><cn>3</cn></apply>
    <ci>a</ci>
  </matrixrow>
  <matrixrow>
    <ci>&beta;</ci><cn>4</cn><cn>5</cn>
  </matrixrow>
</matrix></apply></math>"
  height=75 width=200
```

The way this works is that when techexplorer is installed it is registered to handle data with MIME type `text/mathml`, and the `<embed>` tag accepts attributes that are specific to the plug-in for the specified MIME type, in this example the `mmldata` attribute.

But there are several problems with this. One problem is that the MathML tags may themselves contain attributes with quoted values (e.g. `<cn type="integer">`), although no MathML tags have attributes in the simple example shown here). Fortunately, the flexibility of HTML (and XML) allows the math element to be unambiguously enclosed in single quotes, provided only double quotes are used within the math element. This is the solution that I adopt, since the REDUCE MathML package itself generates only double quotes. An alternative, used automatically by the REDUCE MathML package when the `web` switch is turned on (which I do not use), is to encode the double quotes around MathML attribute values using the SGML entity `"`; . Some reliable technique to avoid quote confusion is *essential* for this inline MathML technique to work.

Another problem is how to determine appropriate values for the `height` and `width` attributes, both of which appear to be *required*. With a static document it should be fairly easy to determine these parameters by trial-and-error, but with a dynamically generated document the only way to determine the display size of a math element is to format it! However, there is a work-around. The width is easy: for displayed (as opposed to text) mathematics it can just be set to 100%, which causes techexplorer to centre the display nicely. One solution for the height is to set it to a value that is large enough in most cases. A value that seems to work in practice and makes sense if the displayed mathematics is the last element on a page, which normally it is in this

application, is also to set the height to 100%. However, after some experimentation I currently set the height to 30%, so that when my demo generates a full transcript (which is an output option) the sign-off message is visible without scrolling a reasonably sized browser window.

Yet another problem is that the `type` attribute is a Netscape extension that may not be supported by other browsers, especially older versions. Under Windows NT 4.0, the above example (embedded in a minimal HTML document) works well in Netscape Navigator 4.7 but in Internet Explorer it causes techexplorer parsing errors. The HTML 4 standard analogue [37] for the `<embed>` tag is the `<object>` tag, but of course this will not be supported by older browsers. Using an `<object>` tag, the above example would look something like this:

```
<object type="text/mathml" height=75 width=200>
<param name="mmldata"
  value="<math><apply><transpose/><matrix>
  <matrixrow>
    <cn>2</cn><apply><root/><cn>3</cn></apply>
    <ci>a</ci>
  </matrixrow>
  <matrixrow>
    <ci>&beta;</ci><cn>4</cn><cn>5</cn>
  </matrixrow>
</matrix></apply></math>">
This browser does not support the object tag!
</object>
```

This version also works well in Netscape Navigator 4.7 but in Internet Explorer it does not display the MathML element at all.

6 HTML, XML and XHTML

HTML is in the process of being supplanted by XHTML (eXtensible Hypertext Markup Language), which is HTML recast as an instance of XML. Development of HTML stopped with the HTML 4.01 standard, which was released on 24 December 1999, and XHTML 1.0 became a W3C Recommendation on 26 January 2000. Fortunately, it can be put to immediate use with existing browsers by following a few simple guidelines. Clearly, it makes sense for any new application to use XHTML rather than HTML, and in the context of embedded MathML this is even more important, since MathML is itself an instance of XML rather than HTML.

At present, there is no official standard way to include MathML within HTML, and it appears that the intention is only to be able to include it in XHTML (but probably never cleanly in HTML). However, the example in the XHTML documentation [11] of including MathML within XHTML does not currently work with my versions of any of Netscape, Internet Explorer, Amaya or Mozilla.

Dave Raggett [12] has given a nice introduction [13] to XHTML. XML is a descendant of SGML, which is easier to implement. The main difference between HTML and XHTML is that the syntax of XHTML is a little more rigid than that of HTML, whilst remaining compatible with HTML, so easing the transition. In particular, XML requires that:

- tags be case-sensitive;
- end tags be included, e.g. `</p>` and ``;

- a / be added to empty tags, e.g. `
` and `<hr />`;
- all attribute values be quoted, e.g. ``.

XHTML uses lower-case for tags and attributes.

In addition to these requirements, a well-written HTML 4.01 document will be valid XHTML 1.0 subject in principle to two provisos. One is that it must begin with an appropriate doctype definition; for the time being, the most appropriate is probably the transitional doctype declaration

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
```

The other is that it must include an XHTML 1.0 namespace as the value of the `xmlns` attribute in the opening `html` tag:

```
<html xmlns="http://www.w3.org/TR/xhtml1">
```

But, in practice, browsers do not seem to care about either of these requirements at present.

A facility that eases the transition from HTML to XHTML is “HTML Tidy” [14]. This is a popular W3C Open Source utility also written by Dave Raggett for fixing up HTML and is available on most platforms. But unfortunately, it does not seem to accept math elements at present.

7 Computer Algebra via CGI

Judging by questions that are continually asked on computer algebra mailing lists, it is worth *outlining* how to set up an HTML form interface to a computer algebra system (CAS). I will phrase this in terms of REDUCE, but the approach is general and I use it also for Maple. Everything hinges on the Common Gateway Interface (CGI) [3]; for further details see also [31] and [36].

When the user selects form elements or enters text into text boxes, the browser sets corresponding variables, and when the user submits the form the browser sends the values of all the form variables to the web server, together with the name of a program to execute. The server then calls the program and passes the form data to it. Most of the messy CGI detail can be hidden by using a WYSIWYG HTML editor to construct the page containing the form and a “standard” library to obtain the form data from within the CGI program.

The CGI program can be implemented in almost any programming language, and can be either a free-standing executable program or a script to be interpreted by an appropriate processor. I use perl [35], which falls into the latter category. It has the advantages of very good string processing, good portability and a lot of support facilities for tasks such as CGI processing.

The CGI program must perform the following essential tasks:

1. get the form data and check its validity;
2. output an appropriate http header;
3. convert the input problem into a form acceptable to the CAS;
4. run the CAS;

5. convert the CAS output into a form that can be displayed by the user’s browser and output it.

It is tempting to consider running the CAS continuously in the background and communicating problems to it as they are received, but this would lead to serious difficulties in keeping the computational environments for different problems separate. It is much easier and cleaner to run a new instance of the CAS for each problem. The overhead in doing this is probably balanced by that of trying to clean up between problems, and avoids an unnecessary background load if problems are run infrequently.

Among several possible ways for the CGI program to communicate with the CAS, two obvious ones are pipes and intermediate files. They have the advantage over mechanisms such as sockets that they do not require any special support from the CAS, and CASs tend not to advertise socket support even if they do in fact provide it. Between pipes and intermediate files, pipes are clearly more elegant, but files are more portable and aid debugging, because they can be left in the filestore for inspection during development. Hence, at present I use temporary files exclusively.

Thus, the CGI program writes the converted form data to a temporary input file. It starts the CAS using an appropriate operating system call, typically “system”, with a command line that causes the CAS to read its input from the temporary input file and write its output to a temporary output file. The CGI program then reads the temporary output file, converts the CAS output and writes it to standard output, which the web server sends back to the user’s browser. The browser displays it according to the information in the http header. After the output has been sent to the browser, the CGI program can delete the two temporary files (or leave them for inspection during development).

One disadvantage of using temporary files concerns the need to distinguish the files belonging to multiple form submissions. One crude temporary solution would be to set the web browser to accept only one connection at a time. However, the solution that I currently use is to attempt to generate a pair of unique filenames, for which I use essentially the time of day to the nearest second. This seems to work well in practice, at least with a lightly loaded server. If necessary, it could be improved by using greater accuracy in the time, a pseudo-random component, or perhaps both.

8 Some Details of the Perl CGI Program

I currently use ActivePerl [32] Build 522 under Windows NT, and the ActiveState Graphical Perl Debugger, for developing my perl CGI programs, and I use the perl CGI.pm module written by Lincoln D. Stein to generate dynamic documents. It is expedient rather than necessary to use such a module, although this module is very well documented and therefore quite easy to use.

The latest distributed version of CGI.pm, version 2.68, available from [33], provides the pragma `-xhtml` for which the documentation is as follows:

```
Enable support for XHTML
(http://www.w3.org/TR/xhtml1/) output. This
changes the default DTD and adds a closing / to
all unpaired tags, such as <BR />.
```

This doesn’t yet (May 2000) make the document fully XHTML compliant, because the tags are still in all-caps.

[I am grateful to Lincoln Stein for drawing my attention to this latest released version via an email exchange.]

It makes sense to generate HTML output that is compatible with XHTML as far as (easily) possible. To then generate true XHTML containing MathML it is simply necessary to omit the `<embed>` markup. For now, I assume that CGI.pm will generate lower-case markup by the time that any browser is available that renders XHTML containing MathML. If necessary, it would not be hard to convert the upper case HTML to lower case explicitly before outputting it.

The demo input form offers the output format options shown in the screen-shot segment in Figure 1, which shows the default settings. Some of the perl script that implements

Output format

Return a *transcript* of the REDUCE session showing *input* as well as output.

Plain text.

The following special formats are at present intended primarily for use with the [IBM techexplorer](#) or equivalent plug-in:

T_EX; MathML *only (no transcript)*;

HTML with embedded MathML;

XHTML containing MathML
(experimental).

Figure 1: Output format options showing the default settings

these options follows, interspersed with a brief commentary. Note that the character `#` introduces a comment in perl; otherwise, perl syntax is roughly C-like. I have had to reformat the perl source code somewhat to fit the layout of this paper; a `\` at the end of a line means that both it and the end-of-line should be ignored.

```
#!/usr/local/bin/perl

# CGI code for CATHODE ODEsolve demo
# Author: Francis J. Wright

use CGI qw(-xhtml); # generate XHTML
$query = new CGI;

# Die to stdout! (STILL NECESSARY?)
sub Die { print @_; exit; }

$Format = $query->param('Format');

if ($Format eq 'Plain') {
    print $query->header('text/plain');
} elsif ($TeX = ($Format eq 'TeX')) {
    # rest for techexplorer
    print $query->header('application/x-tex');
} elsif ($MathML = ($Format eq 'MathML')) {
    print $query->header('text/mathml');
} elsif ($HTML = ($Format eq 'HTML') ||
    ($XHTML = ($Format eq 'XHTML')) {
```

```
    print $query->header(); # default is text/html
} else {
    Die "Invalid output format.\n"
}
```

```
$Transcript = (not $MathML and
    $query->param('Transcript'));
```

The first line is not necessary under Windows, but it allows “-d” to be appended to start the perl debugger, which is extremely useful! Other perl startup options could also be added here if necessary. The subroutine “Die” was introduced before I began using CGI.pm and may no longer be necessary.

The main purpose of this code segment is to load and initialize the perl CGI module and read the values of the form variables `Format` and `Transcript` set by the form segment shown in Figure 1 into the perl variables `$Format` and `$Transcript`. It sets perl “boolean” variables (which default to false) to record the required output format more conveniently and outputs the appropriate http header for the output format selected. It also shows the general mechanism used for input validation. Validating this particular input is just “defensive programming”, since the possible values are constrained by my HTML form, but most other input values could have genuine user errors. A transcript is not possible using only MathML output, because the transcript always contains text (as well as mathematics). In the case of non-transcript MathML-only output, the non-mathematical text is filtered out.

The next code segment processes the problem input, which I will omit since it is not directly relevant to generating MathML output.

```
# Construct the REDUCE input script:

# Directory for temporary REDUCE input and output
# files. (Avoid trying to use pipes for now!)
chdir("/tmp") or
    Die "Can't change to /tmp directory: $!\n";

# Must support multiple simultaneous executions,
# which requires unique filenames. This is a
# quick hack that should normally work. $^T
# returns the time when the program started (in
# seconds since Jan 1, 1970)
$red_in = 'red_in_' . $^T;
$red_out = 'red_out_' . $^T;

open(RED_IN, "> $red_in") or
    Die "Can't open red_in: $!\n";
print RED_IN "off int;\n"; # use batch mode
print RED_IN "remd 'system$\n";
    # to protect against hackers!
print RED_IN "on echo;\n" if $Transcript;
    # display input with output
print RED_IN "load_package $Package;\n";
print RED_IN "fancy_print_df := total$\n"
    if $TeX; # ODEs!
print RED_IN "load_package fmprint; \
on fancy, fancy_tex;\n" if $TeX;
print RED_IN "load_package mathml; on mathml;\n"
    if $MathML || $HTML;
...
print RED_IN "bye;\n";
close RED_IN;
```

The segment above shows how the input file is generated. It contains a complete set of essentially batch-mode REDUCE commands to solve the specified problem. It begins by loading the appropriate solver and then sets up the required output format. In particular, if MathML output is required then the REDUCE MathML package is loaded and turned on. The “...” represents code to write the actual call to the ODE solver into the input file, which I have omitted for brevity.

```
# Run REDUCE:
# *MUST* set 'reduce' environment variable!
$ENV{"reduce"} = $reduce = "E:\\Reduce";
$platform_dir = "$reduce\\lisp\\psl\\win32";
$run_reduce = "$platform_dir\\psl\\bpsl \
-td 12000000 -f $platform_dir\\red\\reduce.img";
system("start /wait /min $run_reduce \
-i $red_in -o $red_out > NUL");
# Note that the "> NUL" is to avoid PSL
# script-mode messages.
```

The next step is to actually run REDUCE. This segment of code is somewhat specific to my server, which runs Windows NT 4.0. For the web demos I run the PSL version of REDUCE (which runs compiled “production” code faster than CSL-REDUCE). It is very easy to modify this code for other platforms, and at one time I ran exactly the same code on both Windows and Solaris by using the fact that a perl script can detect the platform on which it is running. I have omitted the code to check the return status from REDUCE.

```
sub tidy_print {
    my $cond = $_[0];
    # 1 for /<math>/; 0 otherwise
    while (($_ = <RED_OUT>) and
        not ($cond and /<math>/)) {
        if (/^$/ or
            # skip prompts alone and PSL messages
            (not /^(\\d+): *(\\*\\*\\* \\
enlarging fast space by 25000 items)?$/ and
            ($Transcript or not /<Quitting>/))) {
            # skip final 'Quitting'
            s/^(\\d+): *//; # delete any prompts
            print;
        }
    }
}
```

```
#####
```

```
# Post-process the REDUCE output script:
```

```
open(RED_OUT, "$red_out") or
    Die "Can't open red_out: $!\n";

print $query->start_html
    ('Solution of Ordinary Differential Equation')
    if ($HTML);

unless ($MathML) {
    if ($Transcript) {
        print "Transcript using ";
    } else {
        print "Output from ";
    }
}
```

```
while (($_ = <RED_OUT>) and not /^REDUCE/) {
    # skip header junk
    print;
}

if ($TeX) {
    ...
} elsif ($MathML) { # MathML only
    while (($_ = <RED_OUT>) and not /^<math>/) {
        # skip header junk
        print;
        while (($_ = <RED_OUT>) and
            not /^<\/math>/) {print};
        print;
    } elsif ($HTML) { # HTML with embedded MathML
        # Output pre-formatted text:
        print "<pre>\n"; # begin pre-formatted text
        tidy_print 1;
        print "</pre>\n"; # end pre-formatted text
        # Output MathML element. Since it may CONTAIN
        # double-quoted data, MUST enclose the data in
        # SINGLE quotes!
        print "<embed type='text/mathml' mmldata='\"
        unless $XHTML;
        print;
        while (($_ = <RED_OUT>) and not /^<\/math>/)
            {print};
        print;
        print "' height='30%' width='100%'>\n\"
        unless $XHTML;
        print "<pre>\n"; # begin pre-formatted text
        tidy_print 0;
        print "</pre>\n"; # end pre-formatted text
        print $query->end_html;
    } else {
        tidy_print 0;
    }
}

close RED_OUT;

# Delete (unlink) temp files?
unlink $red_in, $red_out;
```

The final code segment above reads the output file, processes its contents and writes the result to standard output, from whence the server sends it to the browser. I have omitted the code that processes T_EX output, part of which I have indicated by “...”.

This code is complicated by several factors. One is the support for several output formats. Another is the fact that PSL-REDUCE may output messages about its internal status, some of which I do not want to pass on to the user and some of which I do want to pass on if the user has requested a transcript. Mainly, the problem is to parse the output and add HTML markup in the right places. I currently output most of the text as pre-formatted elements, so as to preserve the normal REDUCE formatting (mainly line breaking). Then, in order to drive techexplorer, it is necessary to wrap <embed> markup around the math element, although this should not be necessary when generating XHTML.

The very last step is to delete the temporary files. For debugging, I just comment out this line, so that I can read the precise REDUCE input and output generated by a particular problem.

9 Problems

The transcript option on the demo, and the ability to return output in several formats, are both very useful for debugging, and not only of my own code. When the ODE solver fails to solve the ODE, it returns the original ODE (in a slightly modified form). This showed up a couple of problems using MathML output, neither of which arises using \TeX output.

Firstly, the current version of the REDUCE MathML package generates incorrect markup for derivatives. Secondly, techexplorer 2.5.2 renders multiple derivatives incorrectly. I have proposed (and use) a fix for the REDUCE MathML bug, and reported the techexplorer problem to IBM, which I understand will be fixed in the next release.

10 Output Examples

Figures 2 and 3 show two example screen shots using techexplorer within Netscape Navigator 4.7 with the “HTML with embedded MathML” output format. In both cases, I have selected transcript output to show the problem to be solved. Figure 2 shows an ODE that this demo can currently solve, whereas Figure 3 shows one that it cannot, in which case the solver returns the original ODE (in a slightly modified form, namely expanded and collected on the left of the equal sign). In all cases, the solver returns the solution in the form of a list, which techexplorer renders in square brackets (although the REDUCE syntax for a list is curly braces, which might be slightly confusing).

To produce these figures, I made the browser window as small as possible without producing any scroll bars. Figure 3 illustrates the techexplorer 2.5.2 bug in the rendering of multiple derivatives.

11 Future Developments

Several improvements could be made to the demo, some of which may be implemented by the time anyone reads this paper, although they are not all directly relevant to the use of MathML.

The demo page would benefit from the addition of some JavaScript (ECMAScript) to improve and automate the user interface. For example, the browser in use and the availability of suitable plug-ins could be detected and the best available output format selected automatically, or at least made the default. Also, the input could be at least partially validated by the browser. Being able to submit the query by double-clicking on appropriate widgets (e.g. the radio buttons used to select example ODEs) would save the user having to scroll to find an explicit submit button. Event handling in HTML 4.0 is considerably improved from what was readily available in browsers when I developed the first version of this demo, and handling double-click events should now be easy.

The ability to split a browser window into an input and output frame would be better than having the output always appear in the same or a new window. The existing partial support for this needs to be made to work reliably. Previously, it proved difficult to do this across the range of browsers in common use.

In preparation for the next generation of browsers, the demo web page should be converted to XHTML and the CGI program should generate output consisting of XHTML

containing MathML. As far as the MathML is concerned, this just means not generating the markup currently used to embed it. However, it may require a different way of generating markup in the CGI program, because the CGI.pm module that I use at present generates HTML that is not fully XHTML-compliant. If an updated or alternative module is not available then one solution is to code the markup explicitly, which would be easy for this application, although perhaps inelegant.

The fact that the static demo page is currently not XHTML-compliant may be the reason why it seems to crash Amaya. However, Amaya currently supports only MathML presentation markup,² whereas the REDUCE MathML package generates only content markup, so it may still not be possible to use Amaya even if all markup is strict XHTML until it supports MathML content markup. The possibility of using the current version of Mozilla requires the ability to generate XML containing MathML presentation markup. Generating XML should not be too hard, although it seems likely that Mozilla will eventually accept XHTML containing MathML. But the ability to generate *presentation* MathML is clearly desirable.

It would be easy to generalize the processing technique to support more than one segment of MathML, to add more HTML markup, etc., although neither seems appropriate for this demo.

One exciting potential use of MathML *content* markup is for automatic program-to-program communication. For example, one might have a network of solvers for particular classes of ODE that accept input and generate output in MathML content markup. These solvers could be called in the background by an interface that interacts directly with the user. This is an application where OpenMath would be better in principle. But if MathML output is developed primarily for display purposes, then provided it is (or includes) content markup, it could easily be appropriated for program-to-program communication without any further work. Ready availability of good-quality MathML browsers may therefore give MathML an edge over OpenMath.

12 Conclusions

The possibilities to make real practical use of MathML at present are limited. One is constrained to using tortuous methods to embed MathML in documents, and it is much easier and cleaner to just use a \TeX document. The bugs in both the generation and the rendering of MathML mentioned above further indicate the current fragility of the use of MathML in practice. However, I believe that this will change rapidly. We are currently in a chicken-and-egg phase: browser support is limited because there are very few applications because browser support is limited Nevertheless, it is possible to make a tenuous start, as I hope I have indicated, and I expect developments to avalanche.

Acknowledgement

I thank Arrigo Triulzi for his invaluable help with many aspects of computer technology and for his comments on a draft of this paper.

²From the Amaya documentation [8]: “Current limitations: Amaya implements only the Presentation Tags from MathML 2.0, not the Content Markup.”

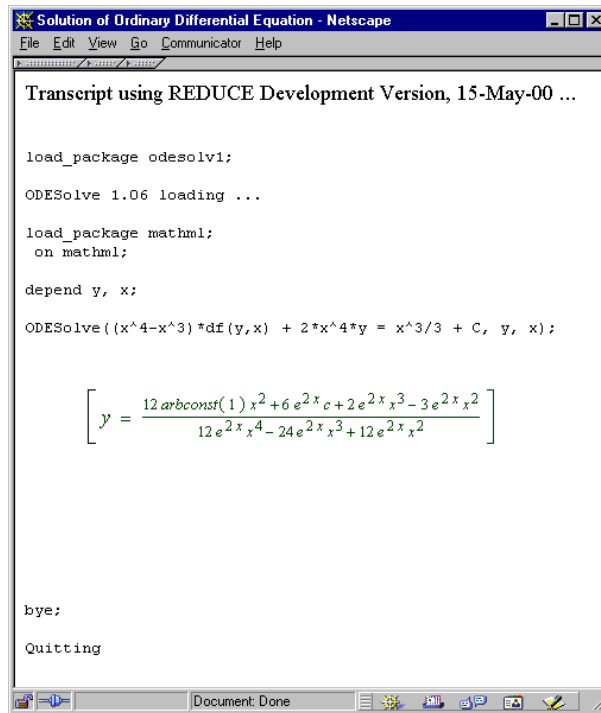


Figure 2: Output example where solution succeeds

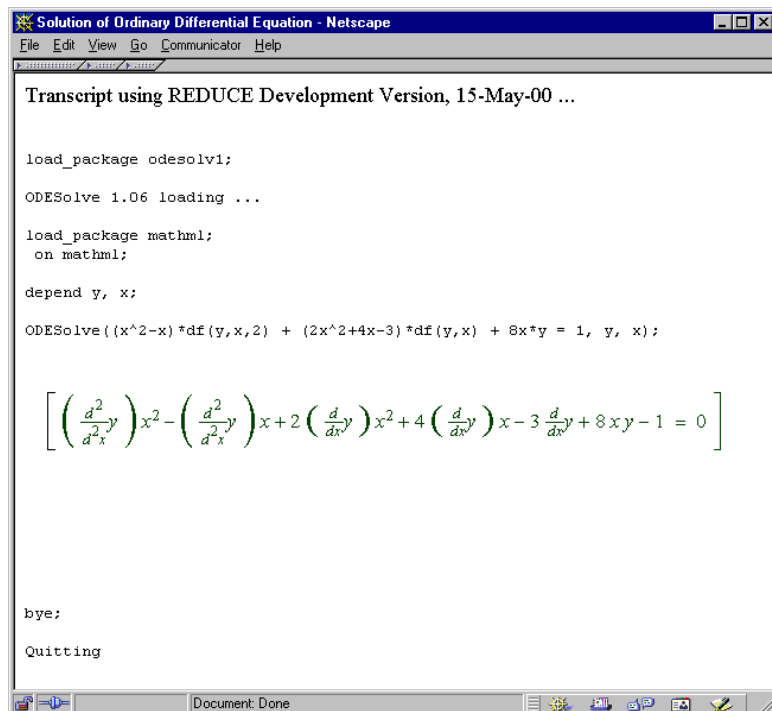


Figure 3: Output example where solution fails

The URLs in the references below all implicitly use the http protocol; I have omitted the http:// prefix for brevity.

References

- [1] www.w3.org/
- [2] www.w3.org/MarkUp/
- [3] www.w3.org/CGI/
- [4] www.w3.org/Math/
- [5] www.w3.org/Math/#Software
- [6] www.w3.org/Amaya/
- [7] www.w3.org/Amaya/MathExamples.html
- [8] www.w3.org/Amaya/User/Math.html#Math
- [9] www.w3.org/TR/REC-MathML/
- [10] www.w3.org/TR/REC-MathML/chapter7.html
- [11] www.w3.org/TR/xhtml1/
- [12] www.w3.org/People/Raggett
- [13] www.w3.org/Talks/1999/12/XHTML-XML99/. See also webreview.com/wr/pub/1999/07/16/feature/index.html, which is additionally available as a single XHTML document webreview.com/1999/07/16/feature/xhtmlonly.html.
- [14] www.w3.org/Talks/1999/12/XHTML-XML99/slide10.html, www.w3.org/People/Raggett/tidy/
- [15] www.openmath.org/
- [16] www-lmc.imag.fr/CATHODE2/
- [17] www.software.ibm.com/techemplorer/; see also centaur.maths.qmw.ac.uk/CTLMathML-Workshop/ for a tutorial.
- [18] www.microsoft.com/windows/ie/
- [19] home.netscape.com/
- [20] www.mozilla.org/
- [21] www.mozilla.org/projects/mathml/
- [22] www.mozilla.org/projects/mathml/update.html
- [23] www.maths.ox.ac.uk/~gartside/mathzilla/
- [24] www.rrz.uni-koeln.de/REDUCE/
- [25] www.zib.de/Symbolik/reduce/moredocs/mathml.pdf
- [26] www.wolfram.com/products/mathematica/newin4/new_publishing.html
- [27] indy.cs.concordia.ca/mathml/software/translators/index.html
- [28] www.gutenberg.eu.org/omega/, genepi.louis-jean.com/omega/lyonmathml.pdf
- [29] www.tug.org/applications/tex4ht/mml/, www.cis.ohio-state.edu/~gurari/TeX4ht/mn.html
- [30] Arrigo Triulzi, OpenMath support under CSL-hosted REDUCE, www.maths.qmw.ac.uk/~arrigo/OpenMath/OpenMath-CSL.pdf. (And of course other papers in this SIGSAM Bulletin OpenMath Special Issue.)
- [31] centaur.maths.qmw.ac.uk/Papers/Marseilles/WebDemo-talk.pdf
- [32] www.ActiveState.com
- [33] CPAN: Comprehensive Perl Archive Network, www.cpan.org/
- [34] *The T_EXbook* by Donald E. Knuth. Addison-Wesley, 1986, ISBN: 0-201-13448-9. *L^AT_EX: A Document Preparation System* by Leslie Lamport. Addison Wesley, 1994, ISBN: 0-201-52983-1. www.tug.org/
- [35] *Programming Perl* by Larry Wall, Tom Christiansen and Randal L. Schwartz. O'Reilly, 2nd Edition, September 1996, ISBN: 1-56592-149-6 [3rd Edition by Larry Wall, Tom Christiansen and Jon Orwant, July 2000 (est.), ISBN: 0-596-00027-8]
- [36] *CGI Programming on the World Wide Web* by Shishir Gundavaram. O'Reilly, 1st Edition, March 1996. *CGI Programming with Perl* by Scott Guelich, Shishir Gundavaram, and Gunther Birznieks. O'Reilly, 2nd Edition, July 2000 (est.), ISBN: 1-56592-419-3.
- [37] *HTML: The Definitive Guide* by Chuck Musciano and Bill Kennedy. O'Reilly, 3rd Edition, August 1998, ISBN: 1-56592-492-4.
- [38] *The L^AT_EX Web Companion: Integrating T_EX, HTML, and XML* by Michel Goossens and Sebastian Rahtz, with Eitan Gurari, Ross Moore and Robert Sutor. Addison-Wesley, 1999, ISBN: 0-201-43311-7.
- [39] More precisely, the OpenMath calls which rely on RPC mechanisms, as defined in Section 9 of the manual, require the remote system to be compiled and set up to receive OpenMath information directly. This appears not to be implemented in any CAS except for an experimental version of REDUCE 3.6. Arrigo Triulzi, private communication.