Introduction to TEX/*Mathematica**

Dan Dill
Chemistry Department, Boston University
Boston MA 02215

September 30, 1993

TEX/*Mathematica* [1, 2] is a set of tools, based on David Jacobson's `math.el` [3], that provide facilities of *Mathematica* Notebooks in a UNIX environment, under GNU Emacs. They permit interaction between a text and a *Mathematica* buffer and, if desired, the use of TEX [4] (or LATEX [5]) to annotate *Mathematica*-based explorations and programs. Inclusion of *Mathematica*-generated graphics in TEX documents printed using PostScript is supported. The tools also support the automatic generation of *Mathematica* packages from TEX/*Mathematica* documents, using a `WEB`-like notation [6, 7].

This report is a tutorial introduction to TEX/*Mathematica*. It shows the different ways you can use TEX/*Mathematica* to interactively develop and refine teaching and research documents, with or without the use of TEX. Thereby *Mathematica*-based exploration can be made a natural part of the writing process.

The tutorial assumes that you have TEX/*Mathematica* installed[1], that you are familiar with GNU Emacs,[2] and, if you want to use TEX or LATEX, that you are familiar with these phototypesetting systems.

1 Annotating Mathematica sessions

Even for simple tasks with *Mathematica*, it is useful to be able to annotate your work as you go along, and to be able to continue your work from one *Mathematica* session to another. This is what TEX/*Mathematica* is designed to allow

 Text of a talk given at the *1992 Mathematica Conference*, Boston, Massachusetts, May 27–31, 1992. Comments and suggestions on this tutorial and on TEX/*Mathematica* itself will be gratefully received, at `dan@chem.bu.edu`.

[1] TEX/*Mathematica* is available by anonymous ftp from chem.bu.edu (128.197.30.18) in `/pub/tex-mathematica`. The files `README` in the subdirectories there describe what to do.

[2] GNU Emacs can be obtained from the Free Software Foundation, 675 Massachusetts Avenue, Cambridge, MA 02139, USA. Readers with access to the Internet can retrieve it by anonymous ftp from `prep.ai.mit.edu`; see the file `/u2/emacs/GETTING.GNU.SOFTWARE` on that host.

you to do. The basic idea is to write your *Mathematica* calculations in a text file, composed of *Mathematica* "cells" together with whatever descriptive text you wish. You can communicate with *Mathematica* from this text file using GNU Emacs. Thereby you can refine and extend the file, always having an up-to-date record of your work, and you can resume working with the text file in subsequent *Mathematica* sessions.

The first step is to start Emacs, say with the UNIX command `emacs tutorial`, and then from within Emacs to start TEX/*Mathematica* with the Emacs command `M-x tex-mathematica`. The Emacs mode line will show `TeX/Mathematica` and the minibuffer will display a message identifying the current version of TEX/*Mathematica*.

The way TEX/*Mathematica* works is by sending *Mathematica* statements that you type in your text buffer (`tutorial` in this example) to *Mathematica* running in a shell buffer, and then by copying the results from *Mathematica* into your text buffer. You identify text meant for *Mathematica* by enclosing it in "cells" delimited as

```
\begin{mathematica}

\end{mathematica}
```

The (Emacs) command `C-c c` (create cell) inserts these delimiters into your text buffer and positions the cursor ready for you to enter *Mathematica* statements into the cell. For example, to compute $\sqrt{\sin(\pi/7)}$, type `Sqrt[ Sin[ Pi/7 ] ] // N`, so that the your text buffer now contains

```
\begin{mathematica}
Sqrt[ Sin[ Pi/7 ] ] // N
\end{mathematica}
```

Once the statement is typed into the cell, you send it to *Mathematica* with the command `ESC RET` (Escape Return). The first time you do this, *Mathematica* is started. Here is what happens. The Emacs display splits into two windows, with the text buffer in one window and a shell buffer in the other window. *Mathematica* is started as a process in the other window in Jacobson's math mode [3]. As soon as the *Mathematica* startup is completed, the text of the cell is sent to *Mathematica*. The *Mathematica* buffer will then look something like

```
Mathematica 2.0 for Sun-3 68881
Copyright 1988-91 Wolfram Research, Inc.
 -- X11 windows graphics initialized --

In[1]:= Sqrt[ Sin[ Pi/7 ] ] // N

Out[1]= 0.658699

In[2]:=
```

You can copy the result into the text buffer with the command `C-c r` (replace). The text buffer will then contain

```
\begin{mathematica}
Sqrt[ Sin[ Pi/7 ] ] // N
.
Out[1]= 0.658699
\end{mathematica}
```

The "`.`" preceding the start of the output is the default TEX/*Mathematica* delimiter; it is easy to change, as illustrated below. These two steps, sending the cell contents to *Mathematica* and then copying the result into the text buffer, can be combined with the single command `C-c u` (update). The update command is also convenient when you make a change in a cell. For example, if instead you wanted $\sqrt{\sin(\pi/8)}$, then you could edit the cell to become

```
\begin{mathematica}
Sqrt[ Sin[ Pi/8 ] ] // N
.
Out[1]= 0.658699
\end{mathematica}
```

and then use `C-c u` to update it to

```
\begin{mathematica}
Sqrt[ Sin[ Pi/8 ] ] // N
.
Out[2]= 0.618614
\end{mathematica}
```

In this way, calculations can be refined as desired.

Cells can contain multiple *Mathematica* statements and a given statement can extend across lines. The only requirements are that multiple statements must be concatenated with "`;`" and that incomplete statements that could be interpreted by *Mathematica* as complete must be continued with "\". For example, the two calculations above could be combined as follows.

```
\begin{mathematica}
arg = {7, 8}; \
x = Sqrt[ Sin[ Pi/# ] ]& /@
    arg // N
.
Out[3]= {0.658699, 0.618614}
\end{mathematica}
```

Here there is no need to indicate continuation in the second line, since `/@` is always followed by an operand and so that line cannot be a complete statement.

The commands that communicate between the text buffer and the *Mathematica* buffer apply only to text within *Mathematica* cells. This means that you can use text outside cells to annotate your work. For example, here is a text buffer containing some TEX/*Mathematica* cells and annotation.

```
This is an example TeX/Mathematica buffer.  Here is a
cell
\begin{mathematica}
1 + 2
\end{mathematica}
and here is another
\begin{mathematica}
Sin[17] // N
\end{mathematica}
We can add additional cells using the command C-c c
(create cell).  We can evaluate cells with ESC RET,
copy Mathematica output into cells with C-c r (replace)
and combine these operations with C-c u (update).

% Local Variables:
% eval: (tex-mathematica)
% eval: (set-variable 'tex-mma-output-marker "-->")
% End:
```

Note that in this example neither cell has been evaluated. The example also illustrates how to use Emacs local variables to invoke TEX/*Mathematica* automatically when the file is edited and also to redefine the default output marker to be "`-->`". Such text files can serve as a record of the *Mathematica* calculations, annotated as you wish.

When starting a new TEX/*Mathematica* session with an existing document, there are several ways you can resume your work. The commands `C-c ]` and `C-c [` move to the next and previous cell in the document, respectively. You can thus use these commands to move to a desired cell and then evaluate it (with `ESC RET` or `C-c u`). The command `C-c a` will optionally evaluate all cells in a document and update their output. Finally, you can specify that only certain cells should be evaluated as a group, analogous to the initialization-cell feature of Notebook front ends. You identify a cell as an initialization cell with the command `C-c TAB`. This changes the cell to

```
\begin{mathematica}[* Initialization Cell *]
...
\end{mathematica}
```

The command `C-c TAB` is a toggle; executing it again converts the cell back into a regular cell. Once you have marked the initialization cells in your document, you can evaluate them as a group with the command `C-c i`.

The table summarizes all you need to use TEX/*Mathematica* to carry out and annotate *Mathematica* calculations from text buffers.

`C-c c`	Create a cell
`ESC RET`	Send a cell to *Mathematica*
`C-c r`	Copy *Mathematica* output into a cell
`C-c u`	Update output portion of a cell
`C-c ]` and `C-c [`	Move forward and backward among cells
`C-c TAB`	Toggle initialization cell
`C-c a` and `C-c i`	Evaluate all or only initialization cells as a group
“`;`” and “\”	For multiple statements and line continuation

Table 1: Basic TeX/*Mathematica* commands

2 Creating Mathematica packages

The interactive nature of TeX/*Mathematica* documents works well with what is probably a typical way of using *Mathematica* to solve problems. Namely, you begin with a general idea of what you want to do, but you are able to determine the details only by trial, error and refinement. In many such cases it becomes clear in retrospect how to encapsulate the solution as a *Mathematica* package so that it can be applied to other problems. TeX/*Mathematica* supports this way of working by allowing you to collect selected cells of a document into *Mathematica* packages.[3]

Here is how this works. You indicate that the text of a cell is to be part of a package by appending a “cell label” of the form <*file*:*part*> to the starting delimiter of the cell, where *file* is the name of the file that you want to contain the package and *part* is the part of the package to which the cell contents belong. The names of the parts of the package are entirely up to you.

You specify how the parts fit together by referring to them in specially labeled cells called “package cells”. Package cells are indicated by a cell label of the form <*file*:>, that is, without a part label. For example, let’s say you want to create a package in the file `MyPackage.m` and a particular cell belongs to the

[3] This feature of TeX/*Mathematica* is modeled on Knuth’s `WEB` system of structured documentation [6, 7]. The basic idea of the `WEB` system is to develop code hierarchically in terms of named fragments. Fragments are introduced as appropriate, in whatever order is convenient to the description of the program, using TeX for documentation. Thereby the author is freed from the constraints of the programming language in the development and description of the code.

In the usual implementation of `WEB` systems, the source document is written in `WEB`, the mixture of code and TeX documentation. The `WEB` document is then alternatively processed by a program `WEAVE` into a file which can then be processed by TeX to give the formatted documentation and code, or by a program `TANGLE` which collects together the code fragments into a source file for the particular language processor (for example, Pascal or C).

In TeX/*Mathematica* instead, since *Mathematica* code is printed verbatim without other special formatting, the TeX/*Mathematica* document itself can be processed directly by TeX (or LaTeX), that is, the `WEAVE` step is not needed. This means that the package generation tools of TeX/*Mathematica* can be used without having to use TeX at all. The generation of the *Mathematica* package code, the `TANGLE` step, is done either interactively from within Emacs or from a UNIX shell.

part of the package that you call `Transformation rules`. You would label this cell as

```
\begin{mathematica}<MyPackage.m:Transformation rules>
...
\end{mathematica}
```

You might have other cells containing usage messages, error messages, utility functions, etc. An example of how you might fit these together into the package `MyPackage` is specified by the package cell

```
\begin{mathematica}<MyPackage.m:>
BeginPackage["MyPackage`"]

<MyPackage.m:Usage messages>

Begin["`Private`"]

<MyPackage.m:Error messages>
<MyPackage.m:Transformation rules>
<MyPackage.m:Utility functions>

End[]; EndPackage[]
\end{mathematica}
```

The blank lines are optional and are inserted for clarity; they will appear in the assembled package.

Cells referenced in package cells can also contain cell references and a particular cell reference can be used multiple times. Cells with the same label will be collated together in the order in which they appear in the document. For example, you might have transformation rules appearing in different cells. You indicate that these are to be combined together in the package simply by labeling each of them as `<MyPackage.m:Transformation rules>`.

Once you have labeled the cells of the package and specified how they are to go together, you can assemble the package in the specified file with the command `C-c m` (make package).

It is useful to make package names and especially part names as descriptive as possible. The command `ESC TAB`, when invoked within an incomplete cell label, will complete file names and part names based on names appearing elsewhere in the document.

The best way to illustrate all of this is with a real example. Before doing that, however, let's see how to use TEX or LATEX to format TEX/*Mathematica* documents, since this tutorial itself is written using LATEX and so it is most natural to use the document formatting features of TEX/*Mathematica* here. In general, however, you do not have to make use of TEX or LATEX to create packages with TEX/*Mathematica*.

3 TeX documents

Once you understand how to use TeX/*Mathematica* to work with *Mathematica* using text files, you can go on and use TeX to format those files, and so you can take advantage of TeX in the text portion of your files. TeX/*Mathematica* provides TeX macros and LaTeX style files to format *Mathematica* cells.

Here are some examples of this formatting. The initialization cell

```
\begin{mathematica}[* Initialization Cell *]
arg = {7, 8}; \
x = Sqrt[ Sin[ Pi/# ] ]& /@
    arg // N
.
Out[3]= {0.658699, 0.618614}
\end{mathematica}
```

looks like when formatted.

```
                                                              [I]
arg = {7, 8}; \
x = Sqrt[ Sin[ Pi/# ] ]& /@
    arg // N
.
Out[72]= {0.658699, 0.618614}
```

The boxed `I` in the cell rule indicates that this is an initialization cell. Here is what the labeled cell

```
\begin{mathematica}<MyPackage.m:Transformation rules>
...
\end{mathematica}
```

looks like,

```
---------- <MyPackage.m:Transformation rules> ----------
...
```

and here is how the package cell

```
\begin{mathematica}<MyPackage.m:>
BeginPackage["MyPackage`"]

<MyPackage.m:Usage messages>

Begin["`Private`"]
```

```
<MyPackage.m:Error messages>
<MyPackage.m:Transformation rules>
<MyPackage.m:Utility functions>

End[]; EndPackage[]
\end{mathematica}
```

appears.

Definition of MyPackage.m

```
BeginPackage["MyPackage`"]

<MyPackage.m:Usage messages>

Begin["`Private`"]

<MyPackage.m:Error messages>
<MyPackage.m:Transformation rules>
<MyPackage.m:Utility functions>

End[]; EndPackage[]
```

A convenient way to use the TEX capability of TEX/*Mathematica* is to start with a template document. Here is a simple example of such a template for TEX,

```
%---TeX-start-of-header---
\input mathematica10pt
\input psfig
\psfiginit
%---TeX-end-of-header---
%---Text starts here---

% Local Variables:
% eval: (tex-mathematica)
% eval: (set-variable (quote TeX-directory) default-directory)
% eval: (setq TeX-start-of-header "%---TeX-start-of-header---")
% eval: (setq TeX-end-of-header "%---TeX-end-of-header---")
% End:
```

and here is an equivalent LATEX template,

```
%---TeX-start-of-header---
\documentstyle[mathematica,psfig]{article}
\begin{document}
\psfiginit
%---TeX-end-of-header---
%---Text starts here---
```

```
\end{document}
% Local Variables:
% eval: (latex-mathematica)
% eval: (set-variable (quote TeX-directory) default-directory)
% eval: (setq TeX-start-of-header "%---TeX-start-of-header---")
% eval: (setq TeX-end-of-header "%---TeX-end-of-header---")
% End:
```

In these templates, the references to `psfig` and `psfiginit` are needed only if you intend to include *Mathematica* graphics. Additional Emacs local variables are set to make invoking TeX or LaTeX from within Emacs convenient: `C-c C-b` processes the buffer through TeX or LaTeX; `C-c C-r` processes a region; `C-c C-p` processes a `dvi` file to a printer; and the current directory is used for intermediate and temporary files. If you have an existing TeX/*Mathematic*-a file that you want to format with TeX or LaTeX you can simply add these statements to the start and end of your file.

4 Graphics

If you *do* use TeX or LaTeX to format your TeX/*Mathematica* text files, then you can incorporate graphics generated by *Mathematica*, using Tomas Rokicki's `dvi` file processor `dvips`[4] with Trevor Darrell's macro package `psfig`, included with TeX/*Mathematica*. First you generate the graphics object in a compressed PostScript file, with the *Mathematica* command `PSTeX[-graphics-, "file"]`.[5] For example,

```
g = Plot[ t^2 E^-t, {t, 0, 10}, Frame -> True ]; \
PSTeX[g, "tutorial-fig"]
.
PSTeX::file: Graphics being processed (without prolog) to file
    "tutorial-fig.ps.Z".

Out[73]= -Graphics-
```

The *Mathematica* prolog is *not* included in the file; rather, the prolog is sent to the printer by the `psfigninit` macro at the beginning of the document. The compressed PostScript file `tutorial-fig.ps.Z` is automatically uncompressed by `dvips`.

[4] The program `dvips` is available by anonymous ftp from `labrea.stanford.edu` as the files `/pub/dvips*`.

[5] You can generate uncompressed files with `PSTeX` by setting `PSTeXCompress` to `False`. In either case, the PostScript bounding box information in the graphics file is duplicated in a file of the same name but with suffix `.bb`. You can also use the shell script `tex-mma-compress` to compress graphics files produced by `PSTeX` and generate the associated `.bb` files.

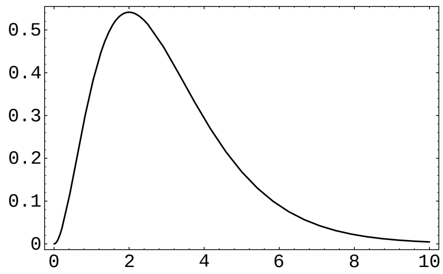

Figure 1: The function $t^2 e^{-t}$.

Once the graphics file has been created, then you can place it in your document using the `psfig` macro. For example, the LaTeX commands

```
\begin{figure}
\centerline{\psfig{figure=tutorial-fig.ps}}
\caption{The function \(t^2 e^{-t}\).}
\end{figure}
```

produce the figure shown here.

5 An example of package creation

Now that we have seen how to format documents, we can develop an example of package creation using LaTeX to format things. Remember, however, that such formatting is not required, but is only for convenience in the presentation of this document.

The example constructs the package `ReIm.m` on page 43 of Maeder's book on programming in *Mathematica* [8], except the package is called `tutorial` here. The package defines rules for the functions `Re[]` and `Im[]` to allow simplification of symbolic expressions containing these functions.

First, the functions need to be unprotected inside the package,

```
<tutorial.m:unprotect>

protected = Unprotect[Re, Im]
.
Out[4]= {Re, Im}
```

and then reprotected at the end of the package.

```
<tutorial.m:protect>

Protect[ Evaluate[ protected ] ]
.
Out[5]= {}
```

The first simplification rule simplifies `Re[x]` to `x` if `x` is real.

```
<tutorial.m:rules>
Re[x_] := x /; Im[x] == 0
```

Simplification rules for addition are

```
<tutorial.m:rules>
Re[x_ + y_] := Re[x] + Re[y]; \
Im[x_ + y_] := Im[x] + Im[y]
```

and for multiplication are

```
<tutorial.m:rules>
Re[x_ y_] := Re[x] Re[y] - Im[x] Im[y]; \
Im[x_ y_] := Re[x] Im[y] + Im[x] Re[y]
```

Here is how the parts of the package fit together.

```
Definition of tutorial.m
BeginPackage["tutorial`"]

Begin["`Private`"]

<tutorial.m:unprotect>

<tutorial.m:rules>

<tutorial.m:protect>

End[]; EndPackage[]
```

Finally, the package is assembled into the file `tutorial.m` with the command `C-c m`. Here is a listing[6] of the resulting package file `tutorial.m`.

```
File tutorial.m
BeginPackage["tutorial`"]

Begin["`Private`"]
```

[6]The file listing is produced here using the macro `mathematicafile` included with TEX/*Mathematica*. It takes one argument, the name of the file.

```
protected = Unprotect[Re, Im]

Re[x_] := x /; Im[x] == 0

Re[x_ + y_] := Re[x] + Re[y]; \
Im[x_ + y_] := Im[x] + Im[y]

Re[x_ y_] := Re[x] Re[y] - Im[x] Im[y]; \
Im[x_ y_] := Re[x] Im[y] + Im[x] Re[y]


Protect[ Evaluate[ protected ] ]

End[]; EndPackage[]
```

This example illustrates several features of package creation with TEX/*Mathematica*. First, the parts of the package can appear in any order in the text file; they are collated into the order specified in package cells, with the command `C-c m`. Second, cells with the same label are collated together. Finally, the output portions of cells (if any) are *not* copied into the package; this makes it convenient to debug components of packages before assembly.

All of this makes it easy to construct packages from a text file (more than one package can be specifed in a given text file) and to keep the definition of the package current as you make changes to the text file, by remaking the package with the command `C-c m` as necessary. In this way you can use TEX/*Mathematica* files to provide full details of the design and documentation of *Mathematica* packages.

6 Conclusion

This tutorial has treated the most common TEX/*Mathematica* commands and facilities. There are other specialized features that have not been discussed here. Most notably, the concept of cells and packages is readily extended to languages other than *Mathematica*. For example, by defining a new cell type for C, TEX/*Mathematica* can be used to document the development of C programs. The fragments of C code are assembled into files using the package feature of TEX/*Mathematica*. All of the features of TEX/*Mathematica* are described in the Texinfo document included with TEX/*Mathematica*.

References

[1] Dan Dill. Interactive TEX/*Mathematica* documents. *Mathematica J.*, 1(2):104–107, 1991.

[2] Dan Dill. Creating packages with TEX/*Mathematica*. *Mathematica J.*, 1(4):16–17, 1991.

[3] David M. Jacobson. *Mathematica* Mode for Gnu Emacs. *Mathematica J.*, 1(2):29, 1990.

[4] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984. It is important to get at least the 18th printing, since it contains recent changes to TEX.

[5] Leslie Lamport. LATEX*: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 1986.

[6] Donald E. Knuth. *Literate Programming*. Center for the Study of Language Information, Stanford University, 1992.

[7] Wayne Sewell. *Weaving a Program: Literate Programming in* `WEB`. Van Nostrand Reinhold, New York, 1989.

[8] Roman Maeder. *Programming in Mathematica*. Addison-Wesley, Redwood City, California, 2nd edition, 1991.