The MatTEX package

Romeo Van Snick
romeovs@gmail.com

February 11, 2012

MatTEX is a rudimentary set of LATEX macros and matlab scripts that try to make your life easier when you are working with calculated matlab or octave numerical values in a LATEX document.

In stead of copy-pasting the calculated values into the latex document, it is possible to export them to an intermediary file that can be `\input`'ed into the document. Saving you from having to do the job twice if you changed somethng in the calculations and get a different numerical result.

In what follows I'll try to desccribe how to use the LATEX macro's as well as the matlab scripts that can be used to save the values of you calculated values. But first, a very short introduction into the nomenclature of the scientific notation.

1 About numbers

The standard (as far as I know) way to scientifically write a number is generally:

$$(x \pm s_x) \times 10^e$$

In this notation several parts are distinguised:

- the significand x
- the uncertianty estimate or error s_x
- the exponential e

Usually the uncertainty estimate is rounded off to two *significant digits*. The significand is then rounded to match the rounding of the uncertainty estimate, so that the last two significant digits match the order of magnitude of those of the uncertainty estimate. The exponent is then adapted so that the order of magnitude of the total is still correct.

Please visit `http://www.chemteam.info/SigFigs/SigFigs.html` for a more in depth discussion on the scientific notation.

I will use the above nomenclature to denote the parts of a number. A number, consisting of all of the above parts (or a subset of them), will be denoted by the term *value*. The name you give this number (i.e. the place you store it in) will be called a *variable*.

So if you would type:

```
>> a = 14e3;
```

into a matlab or octave prompt, you will have created a variable called `a` with `value` that is `14e3`. This value is compromised of a significand `14` and an exponent `3`.

To have a variable with a value that also has an error estimate you should type[1]:

```
>> a = [21.3 1.1]*1e3;
```

To get a value that has a significand `12.1`, an exponent `3` and an uncertainty estimate `1.1`.

2 LaTeX usage

This section will discuss the correct usage of the LaTeX macro's that are supplied in this package.

To gain acces to the macro's put the following in the preamble of your document:

```
\usepackage{mattex}
```

This will allow you to use the macro's explained in the following sections.

2.1 Setting variable values using `Mset`

To set a variable, you can use following syntax:

`\Mset{`⟨*name*⟩`}{`⟨*significand*⟩`}{`⟨*error*⟩`}{`⟨*exponent*⟩`}` `\Mset`

This will set the variable named ⟨*name*⟩ to:

$$(\langle \textit{significand} \rangle \pm \langle \textit{error} \rangle) \times 10^{\langle \textit{exponent} \rangle}$$

Setting a variable overwrites it's previous value (if it already had a value).

Some examples:

`\Mset{a}{12}`	set the variable `a` to 12
`\Mset{b}{123}{24}`	set the variable `b` to 123 ± 14
`\Mset{c}{34}{}{-5}`	set the variable `c` to 34×10^{-5}
`\Mset{d}{72}{11}{-9}`	set the variable `d` to $(72 \pm 11) \times 10^{-9}$

The names can be almost anything, from alphabetic letters such as `a`, `b`, ... to numbers `0`, `1`, ... and multicharacter comdinations of these, e.g. `ab10`. Also, underscores, commas and braces are allowed so it's possible to make a variable that is called `a_crit(2,3)`. The only thing I know of that is not allowed are, for obvious reasons, the tex active characters such as `\`, `{`, `}`, `[` and `]`. Note that

[1] This of couse can be done in other ways, yet is the necessary method to add variables and their error in the matlab scripts that join the MatTeX package.

most of these symbols cannot be used for argument names in matlab either, so this doesn't matter that much (if you don't understand this sentence, please ignore it and read on).

2.2 Getting the values

Dependig on shoch parts of the value of a variable you need, you can use different macro's to get them (note that some knowledge of the `siunitx` package is advised while reading this section).

In what follows we will assume a variable was set as follows:

```
\Mset{e}{72}{11}{-9}
```

`\Mval{`⟨*name*⟩`}` `\Mval`

Gives you the significand and exponent of the variable ⟨*name*⟩ as if put through the `\num` macro. So for instance `\Mval{e}` would be equal to typing `\num{72e-9}`. The typeset result should be: 72×10^{-9}.

Here is were the reason d'être of this package lies. You could of course just have typed in `\num{72e-9}` in stead of going trough the hassle of first assigning this value to a variable and then getting it, yet this has a number of disadvantages compared to the MatTEX approach:

- consistency: will the number you enter be entered consistenly troughout the document? Chances are you type a mistake or change notation in the middle of a document.
- what if you made a mistake in the matlab calculations? You'd have to do a lot of work over again, editing each occurence of the variable in the document, possibly missing some occureces.
- you really have to do the work of copy pasting tha value over from matlab yourself. This, to me is perhaps the biggest disadvantage. Using the mattex package, you can let your latex and matlab files coexist in a makefile and compile the whole bunch in one go, without the need of you interviening to copy-paste stuff over.

The name `\Mval` might seem a bit in contradiction to the nomenclature of this manual, but it makes sense: you only want the calculated *value* of a variable[2].

`\Merr{`⟨*name*⟩`}` `\Merr`

Gives you the same as `\Mval` but returns the uncertainty estimate instead, `\Merr{e}` results in 11.

`\Mnum{`⟨*name*⟩`}` `\Mnum`

This gives you the full value of variable ⟨*name*⟩. `\Mnum{d}` would result in $(72 \pm 11) \times 10^{-9}$.

[2] here value denotes significand with exponent.

\MSI{⟨*name*⟩}{⟨*unit*⟩} \MSI

This results in the same as \Mnum but allows you to append a unit specified by ⟨*unit*⟩, as you would do using the \SI command from the siuntix package. The rules that apply for the siunitx package appy here too as well as the settings you may have set for it. \Mnum{d}{\metre} will result in: $(72 \pm 11) \times 10^{-9}$ m. Please also read the siunitx package documentation.

Note: For all of the above commands the same rule applies: if a certain part of a variable isn't set, it wont show up in any of these commands. So if we were to redifine our value d to \Mval{72}{}{-9} (i.e. without an error estimate set), \Mval would remain the same, \Merr would result in literally 'nothing' and \Mnum would be equal to \Mval.

2.3 Internal macro's

You probably won't need the following macro's, yet they might come in handy if you want to define your own stuff based on what MatTEX macro's can do. The strings that come out of the following macro's can be passed on to the siunitx macro's for example (which is basically what is done in the macro's above).

\M{⟨*name*⟩} \M

This gets the literal string that is saved under the variable with name ⟨*name*⟩. This macro returns a simple text string such as 72+-11e-9 or 12e-23.

\Mvallit{⟨*name*⟩} \Mvallit
Gives the literal string that is saved under the significand and exponent of variable ⟨*name*⟩. For \Mset{e}{93}{9}{6} this results in 93e9.

\Merrlit{⟨*name*⟩} \Merrlit

Gives the literal string that is saved under the error with exponent of variable ⟨*name*⟩. For \Mset{f}{100}{10}{-9} this results in 10e-9.

2.4 Matrices

Since commas and braces are allowed in the name of a variable, this can be used to make a matrix with indeces. There is also a matlab script that outputs matlab arrays to variables that MatTEX -readable (see writemat).

Te generate quick matrices use the following commands:

\preparematrix{⟨*name*⟩}{⟨*N*⟩}{⟨*M*⟩} \preparematrix
\usematrix \usematrix

Using the comidination of \preparematrix and \usematrix you can easily build a matrix. Say we have a 2 by 2 matrix called A. We set the elements using:

```
\Mset{A(1,1)}{1}{0.1}
\Mset{A(1,2)}{2}{0.2}
\Mset{A(2,1)}{3}{0.3}
\Mset{A(2,2)}{4}{0.4}
```

If the elements are set using the above approach, we can prepare the matrix for use in a tabular environment using `\preparematrix`. This macro takes 3 argumentsr. The first argument $\langle name \rangle$, which in our case is `A`, the name of the matrix. The remaining arguments $\langle N \rangle$ and $\langle M \rangle$ tell `\preparematrix` that the size of the matrix is: $\langle N \rangle$ is the number of rows and $\langle M \rangle$ is the number of colums.

```
\preparematrixmatrix{A}{2}{2}
\begin{tabular}{cc}
        \noheader
        \hline
        \usematrix
        \hline
\end{tabular}
```

should result in:

A(1,1)	A(1,2)
A(2,1)	A(2,2)

Of course the above table isn't what we'd want to see, we need the values of the elements, not there names. To do this there are three column types defined by MatTEX :

- `v`: sets every cell of that column in an `\Mval` command.
- `n`: sets every cell of that column in an `\Mnum` command.
- `e`: sets every cell of that column in an `\Merr` command.

This allows us to get the values of the table cells:

```
\preparematrixmatrix{A}{2}{2}
\begin{tabular}{vn}
        \noheader
        \hline
        \usematrix
        \hline
\end{tabular}
```

would result in:

1	2.0 ± 0.2
3	4.0 ± 0.4

Now there might be cells you don't want to have put trough the respective `\M...` command when using one of the above column types, header or footer rows

for instance. This is where the `\header` and `\noheader` commands come in.

`\header` `\header`

Starts the header in a `tabular` environment. The cells on each row after the `\header` macro are not put into the `\M...` commands even when the `v`, `n` or `e` columns are used.

`\noheader` `\noheader`

Stops the header in a `tabular` environment. For example:

```
\begin{tabular}{vn}
        \hline \header
        Hdr 1 & Hdr 2 \noheader \\
        \hline
        \usematrix
        \hline
\end{tabular}
```

Which should result in:

Hdr 1	Hdr 2
1	2.0 ± 0.2
3	4.0 ± 0.4

2.5 Examples

This example shows how the output is when all information is set for a variable:

```
\Mset{a}{123}{45}{6} % set a
\Mval{a}                        123 × 10^6
\Merr{a}                        45 × 10^6
\Mnum{a}                        (123 ± 45) × 10^6
\MSI{a}{\kilo\gram}             (123 ± 45) × 10^6 km
\M{a}                           123+-45e6
\Mvallit{a}                     123e6
\Merrlit{a}                     45e6
```

The next example shows how the output is when only the value is set for a variable:

```
\Mset{b}{78} % set b
\Mval{b}                        78
\Merr{b}
\Mnum{b}                        78
\MSI{b}{\kilo\gram}             78 km
\M{b}                           78
\Mvallit{b}                     78
\Merrlit{b}
```

This example shows you how to make a nice table from some values:

```
\Mset{c(1,1)}{12.1}{1.1}
\Mset{c(2,1)}{13.3}{1.0}
\Mset{c(3,1)}{11.2}{0.9}
\Mset{c(4,1)}{11.9}{1.3}
\Mset{c(5,1)}{12.5}{0.8}

\Mset{c(1,2)}{455}{14}
\Mset{c(2,2)}{457}{12}
\Mset{c(3,2)}{453.2}{7.3}
\Mset{c(4,2)}{455}{13}
\Mset{c(5,2)}{458}{12}

\makematrix{c}{5}{2}
\begin{table}[htp]
        \centering
        \begin{tabular}{MM}
                \hline\header
                Head 1 & Head 2 \noheader \\
                \hline
                \usematrix
                \hline \header
                Foot 1 & Foot 2\\
                \hline
        \end{tabular}
\end{table}
```

Should result in:

Head 1	Head 2
12.1 ± 1.1	455 ± 14
13.3 ± 1.0	457 ± 12
11.2 ± 0.9	453.2 ± 7.3
11.9 ± 1.3	455 ± 13
12.5 ± 0.8	458 ± 12
Foot 1	Foot 2

2.6 Requirements

Several packages are needed to be able to use MatTEX , they are listed below:

- `pgfkeys`: this is used to set and save the variables.
- `xstring`: used to compare strings.
- `siunitx`: this one should be obvious. If you need this package, you'll probably have this loaded already.
- `xparse`: for the flexible defining of commands.
- `array`: for the defining of new column types in tabular.
- `collcell`: also used for the special tabular cells.

3 Matlab usage

With MatTEX come several matlab scripts that export variables in the correct format, so that MatTEX can pick them up.

These scripts were initially written in matlab and then tested in octave, so they should work on both. Recently, however, I've stopped using matlab and have switched to octave completely. This, however, should pose no porblem since I come from a matlab background and my syntax hasn't changed, so I expect the scripts to work with matlab as well. The main reason why I switched to octave is that it works as a genuine compiler, so it can be used properly in makefiles etc. This makes the combination latex-mattex-octave extremely powerful. What follows is applicable to octave as well as to matlab.

To use the scripts, make sure that the files `writevars.m`, `writeallvars.m`, `formatvars.m`, `writemat.m` and `parsemopts.m` are in a directory that matlab can read.

`writevars(`⟨*file*⟩`,`⟨*opts*⟩`,`⟨*var1*⟩`,`⟨*var2*⟩`,...)` `writevars`

This function can be used to write variables ⟨*var1*⟩, ⟨*var2*⟩, ... to a file called ⟨*file*⟩. The variables are automatically formatted is such a fashion that the error (if there is one) has got two significant digits and that the significand has corresponding meaningful digits.

Through the optional argument ⟨*opts*⟩ you can specify how the data should be written:

- `a`: append the variables to the file. This is default behaviour so this option does not have to explicitly given.
- `w`: write tto the file instead of appending. This clears the file before writing to it. This option overwrites the default append behaviour.
- `s`: be silent. This causes writevars to refrain from writing information to the prompt and causes it not to write the datestring into the file. This is generally a good idea when writing from a loop.
- `e`: force exponent. This causes numbers that have magnitude -1, 0 or 1 to be written with exponent (eg `1e-1` instead of `0.1` and `1e0` instead of `1`), which normaly would not be done.
- `#`: any number greater than or equal to 1. This number denotes the number of significant digits that will be used for the error (the number of significant digits on the significand will change accordingly).

These options should be put in a `char` string (the order doesn't matter). For instance `'wse4'` will write the values using 4 significant digits (clearing the file first), not print any additional information to the screen and use scientific notation, even when the magnitude is -1, 0 or 1.

The variables *must* be passed by name in matlab, and they will have the same name in the LaTeXfile as the name they were passed by. To give a value with an error pass a vector (also by name) containing the value and error.

writeallvars(⟨*file*⟩,⟨*opts*⟩) writeallvars

This basically does the same as `writevars` yet it searches the workspace for all the current variables that can be written (i.e. `double` and size ≤ 2) and writes them. The same options as for `writevars` apply.

writemat(⟨*file*⟩,⟨*matrix*⟩,⟨*error matrix*⟩) writemat

Writes all the elements of the matlab array ⟨*matrix*⟩ to the file specified in ⟨*file*⟩, with optional errors specified in the array ⟨*error matrix*⟩. After inputting ⟨*file*⟩ has been input into LaTeXthe variables are available trough:

`\Mval{`⟨*matrix*⟩`(`⟨*i*⟩`,`⟨*j*⟩`)}`

For example the value in the third row and second column of a matrix `A` is called by `\Mnum{a(3,2)}`. This matrix is of the correct form to be used in the `\preparematrix` approached described above.

4 Recommended workflow

To get the variables from a matlab session into a LaTeXdocument I recommend using the following workflow:

1. Do the calculations in matlab. You matlab script could look like this:

```
a = [3.4 2.9 3.5 3.1 4.5];
b = mean(a);
s_b = std(a);
c = [b s_b];
d = exp(c)*100000;
```

2. Export the variables to a file `vars.tex`, using one of the above matlab functions. Do this by adding:

```
writevars('file.tex','a',c,d);
```

   The file `file.tex` should look like this after you run your script:

```
\Mset{c}{3.48}{0.62}
\Mset{d}{32.5}{1.9}{5}
```

3. Input the file by putting `\input{file.tex}` in the TeXdocument.

4. Use the variables defined in `file.tex` by using the macro's described above. For instance:

```
\input{file.tex}
\Mnum{c}
\MSI{d}{\metre}
```

   This should result in:

   3.48 ± 0.62
   $(32.5 \pm 1.9) \times 10^5\,\mathrm{m}$

5 Thanks!

A lot of credit goes out to Joseph Wright, the editor of the `siunitx` package. I've been a devote `siunitx` user for years now.

I'm also in debt to the poeple at `http://tex.stackexchange.com/`, who do their best to answer every single one of my questions.

Index

\header, 6

\M, 4
\Merr, 3
\Merrlit, 4
\Mnum, 3
\Mset, 2
\MSI, 4
\Mval, 3
\Mvallit, 4

\noheader, 6

\preparematrix, 4

\usematrix, 4

writeallvars, 9
writemat, 9
writevars, 8