Documentation for matrix2latex

Øystein Bjørndal

November 29, 2013

This is the documentation for the matrix2latex package, development can be found here: `http://code.google.com/p/matrix2latex/`

This package provides easy translation of vector/matrices in python or matlab into latex table/matrix code. The implementation is written in both matlab and python, resulting in some discrepancies in features and usage. After the introduction we will discuss some matlab specific usage in chapter 2 before showing all options using the python syntax. Currently the python version is the best maintained.

Chapter 1

Introduction/features

Takes a python or matlab matrix or nested list and converts to a LaTeX table or matrix. Author: ob@cakebox.net, inspired by the work of koehler@in.tum.de who has written a similar package only for matlab `http://www.mathworks.com/matlabcentral/fileexchange/4894-matrix2latex`

This software is published under the GNU GPL, by the free software foundation. See: `http://www.gnu.org/licenses/licenses.html#GPL`

1.1 TODO

- Provide installation.
- Complete test suite.
- Improve error handling

1.2 Compatibility

Table 1.1 reflect the current features and whether it is available in the matlab or python version. These features are discussed in more detail in chapter 3.

1.2.1 Python

The code is written without any dependencies and should be compatible with most python versions. Table 1.2 reflects the python versions currently installed on my system[1] and if the testsuit for matrix2latex passes. If you find a python version where it doesn't work let me know.

[1] Mac OS X 10.6, python installed trough macport.

Table 1.1: What feature is currently available in which language?

Feature	Python	Matlab
environment	True	True
headerRow	True	True
multiline headerRow	True	False
headerColumn	True	True
multiline headerColumn	False	False
transpose	True	True
caption	True	True
label	True	True
format	True	True
formatColumn	True	True
alignment	True	True

Table 1.2: Does 'python test.py' return 0?

	Compatible
python2.4	True
python2.5	True
python2.6	True
python2.7	True
python3.3	True

1.2.2 Matlab

For the moment it is only tested with matlab R2009b.

1.3 Installation

The following packages and definitions are recommended in the latex preamble

```
% scientific notation, 1\e{9} will print as 1x10^9
\providecommand{\e}[1]{\ensuremath{\times 10^{#1}}}
\usepackage{amsmath} % needed for pmatrix
\usepackage{booktabs} % Fancy tables
\usepackage{caption} % Fixes spacing between caption and table
```

```
...
\begin{document}
...
```

1.3.1 Python

To use the code place the folder matrix2latex/ in our `PYTHONPATH`. Your current working directory is always on your `PYTHONPATH`.

Hint: on unix systems do:
`echo $PYTHONPATH`
to see a list of locations. Other users: ask google about `PYTHONPATH` for your operation system.

Chapter 2

Matlab specific

2.1 Installation

Place the matlab files in our `MATLABPATH`. Your current working directory is always on your `MATLABPATH`.

See the following article from mathworks `http://www.mathworks.se/help/techdoc/ref/path.html`.

2.2 Usage

As matlab does not support keywords, something similar to the environment must be used. Most of the examples in this document are given in python code only. The matlab version should be identical in behavior but has slightly different syntax.

```
matrix2latex(matrix, filename, varargin)
```

```
% Filename must be suppiled but filename == '' will not write to file.
% filename = 'foo.tex' and filename = 'foo' will write to 'foo.tex'
m = [1, 1; 2, 4; 3, 9];
t = matrix2latex(m, '');
% keyword arguments are given like this
% instead of transpose = True
t = matrix2latex(m, '', 'transpose', true);
% environment is a keyword argument
t = matrix2latex(m, '', 'environment', {'align*', 'pmatrix'});
% if you want to pass in strings you must use cell array
m = {'a', 'b', '1'; '1', '2', '3'}
t = matrix2latex(m, '', 'format', '%s');
```

Chapter 3

General Usage

3.1 Usage

The python version should be called like this

```
matrix2latex(matrix, filename, *environment, **keywords)
```

3.1.1 matrix

Python: A numpy matrix, a pandas DataFrame/Series or a (nested) list. Note: these are all converted to a nested list internally for compatibility.

Matlab: Matrix or cell array.

3.1.2 Filename

File to place output, extension .tex is added automatically. File can be included in a LaTeX document by `\input{filename}`. Output will always be returned in a string. If filename is None not a string or empty string it is ignored.

3.1.3 *environments

Use `matrix2latex(m, None, "align*", "pmatrix", ...)` for matrix. This will give

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Use `matrix2latex(m, "test", "table", "center", "tabular" ...)` for table. Table is default so given no arguments: table, center and tabular will

be used. The above command is then equivalent to
`matrix2latex(m, "test", ...)`

The above commands looks a bit differently in matlab, since we must specify that we want to change the environment.

```
matrix2latex(m, None, 'environment', {'align*', 'pmatrix'}, ...)
matrix2latex(m, 'test', 'environment', {'table', 'center', 'tabular'} ...)
matrix2latex(m, 'test', ...)
```

Example

```
from matrix2latex import matrix2latex
m = [[1, 1], [2, 4], [3, 9]] # python nested list
t = matrix2latex(m)
print(t)
```

```
\begin{center}
  \begin{tabular}{cc}
    \toprule
    $1$ & $1$\\
    $2$ & $4$\\
    $3$ & $9$\\
    \bottomrule
  \end{tabular}
\end{center}
\end{table}
```

1	1
2	4
3	9

3.1.4 **keywords

headerRow

A row at the top used to label the columns. Must be a list of strings. Using the same example from above we can add row labels

```
hr = ['$x$', '$x^2$']
t = matrix2latex(m, headerRow=hr)
```

Which in matlab looks like this (note that cell array must be used for declaring the header row)

```
hr = {'$x$', '$x^2$'};
t = matrix2latex(m, 'headerRow', hr);
```

x	x^2
1	1
2	4
3	9

The python version currently supports header rows spanning multiple columns (using \multicolumn), this is done by repeating the header info. It also supports multiple rows by using a nested list.

```
hr = [['Item', 'Item', ''],
      ['Animal', 'Description', 'Price (\$)']]
t = [['Gnat', 'per gram', 13.65],
     ['', 'each',  0.01],
     ['Gnu', 'stuffed', 92.50],
     ['Emu', 'stuffed', 33.33],
     ['Armadillo', 'frozen', 8.99]]
t = matrix2latex(t, headerRow = hr, alignment='@{}llr@{}')
```

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.5
Emu	stuffed	33.33
Armadillo	frozen	8.99

To avoid this behavior ensure each consecutive item is unique, for instance: `['Item', 'Item ', '']` (note the space after the second item).

headerColumn

A column used to label the rows. Must be a list of strings

transpose

Flips the table around in case you messed up. Equivalent to matrix2latex(m.H, ...) if m is a numpy matrix. Note that `headerColumn` is used in the example.

```
t = matrix2latex(m, headerColumn=hr, transpose=True)
```

x	1	2	3
x^2	1	4	9

caption

Use to define a caption for your table. Inserts `\caption` after `\begin{center}`, note that without the center environment the caption is currently ignored. Always use informative captions!

```
t = matrix2latex(m, headerRow=hr,
                 caption='Nice table!')
```

Table 3.1: Nice table!

x	x^2
1	1
2	4
3	9

label

Used to insert `\label{tab:...}` after `\end{tabular}` Default is filename without extension.

We can use `label='niceTable'` but if we save it to file the default label is the filename, so:

```
matrix2latex(m, 'niceTable', headerRow=hr,
                 caption='Nice table!')
```

can be referenced by `\ref{tab:niceTable}`. Table 3.2 was included in latex by `\input{niceTable}`.

Table 3.2: Nice table!

x	x^2
1	1
2	4
3	9

format

Printf syntax format, e.g. `$%.2f$`. Default is `$%g$`. This format is then used for all the elements in the table. Using numpy creating tables for common mathematical expressions are easy:

```
import numpy as np
m = np.zeros((3, 2))
m[:, 0] = np.arange(1, 3+1)
m[:, 1] = 1./m[:, 0]

hr = ['$x$', '$1/x$']
t = matrix2latex(m, headerRow=hr,
                 format='%.2f')
```

x	$1/x$
1.00	1.00
2.00	0.50
3.00	0.33

formatColumn

A list of printf-syntax formats, e.g. `['$%.2f$', '$%g$']` Must be of same length as the number of columns. Format i is then used for column i. This is useful if some of your data should be printed with more significant figures than other parts, for instance in the example above x are integers and using d is more appropriate.

```
fc = ['$%d$', '$%.2f$']
t = matrix2latex(m, headerRow=hr, formatColumn=fc)
```

x	$1/x$
1	1.00
2	0.50
3	0.33

You could use the format option to add units to you numbers, so for instance visualizing ohms law

```
m2 = np.zeros((3, 2))
R = 50. # ohm
m2[:, 0] = np.arange(1, 3+1)
m2[:, 1] = m[:, 0]/R
c = 'Current through $%g \Omega$ resistor' % R
hr = ['$V$', '$I=V/R$']
t = matrix2latex(m2, 'table_ohm', headerRow=hr,
                 formatColumn=['$%d V$', '$%.2f A$'],
                 caption=c)
```

Table 3.3: Current through 50Ω resistor

V	$I = V/R$
$1V$	$0.02A$
$2V$	$0.04A$
$3V$	$0.06A$

This is however not the recommend way to give units, since they should be given in the header, see tabel 3.4.

```
hr = ['$V$ [V]', '$I=V/R$ [A]']
t = matrix2latex(m2, 'table_ohm2', headerRow=hr,
                 formatColumn=['$%d$', '$%.2f$'],
                 caption=c)
```

alignment

Used as an option when tabular is given as enviroment. `\begin{tabular}{alignment}` A latex alignment like c, l or r. Can be given either as one per column e.g.

Table 3.4: Current through 50Ω resistor

V [V]	$I = V/R$ [A]
1	0.02
2	0.04
3	0.06

"ccc". Or if only a single character is given e.g. "c", it will produce the correct amount depending on the number of columns. Default is "c", if you use `headerColumn` it will always use "r" as the alignment for that column.

```
t = matrix2latex(m, alignment='rl')
```

1	1
2	0.5
3	0.333333

But what if I want vertical rules in my table? Well, this package is built on top of booktabs for publication ready tables and the booktabs documentation clearly states "Never, ever use vertical rule". But as long as you are not publishing your table, you could simply use

```
t = matrix2latex(m, alignment='|r||c|')
```

1	1
2	0.5
3	0.333333

There is some error checking on the alignment but not much, it simply counts the number of c, l and r in the alignment. All other characters are ignored.

position

Used for the table environment to specify the optional parameter "position specifier" Default is `'[' + 'htp' + ']'`

If you want to place your table manually, do not use the table environment.

General considerations

Note that many of these options only has an effect when typesetting a table, if the correct environment is not given the arguments are simply ignored. To give an example of a very useless function call

```
t1 = matrix2latex(m, None, "align*", "pmatrix",
                  alignment='rc',
                  caption='hello world',
                  label='test')
# produces the exact same thing as
t2 = matrix2latex(m, None, "align*", "pmatrix")
assert t1 == t2
```

$$\begin{pmatrix} 1 & 1 \\ 2 & 0.5 \\ 3 & 0.333333 \end{pmatrix}$$

The scary thing is that `headerColumn` actually works when creating a matrix, it just looks a bit wierd.

The options presented by this program represents what I need when creating a table, if you need a more sophisticated table you must either change the python code (feel free to submit a patch), manually adjust the output afterwards or adjust the input (remember that the input can be a nested list of strings). `http://en.wikibooks.org/wiki/LaTeX/Tables` gives an excellent overview of some advanced table techniques.

The booktabs.pdf documentation is an excellent guide to proper table creation, matrix2latex does not incorporate all the features of this package (yet).

3.1.5 Pandas support

Some preliminary support for Pandas[1] is available, some examples:

```
import pandas as pd
s = pd.Series([2, 4, 2, 42, 5], index=['a', 'b', 'c', 'd', 'e'])
t = matrix2latex(s)
```

[1] `http://pandas.pydata.org/`

2
4
2
42
5

3.2 Usage examples

The usefulness of a programming interface to create LaTeX tables becomes apparent when the data is dynamically created by python. This can be either because you want flexibility with respect to the tables size or because the table content is somehow created by python.

One day you decide to compare different implementations of the factorial functions, you start by writing the following file as `factorial.py`

```
# built in factorial
from math import factorial as factorialMath

# recursive
def factorialRecursive(n):
    if n == 0:
        return 1
    elif n == 1:
        return n
    else:
        return n*factorialRecursive(n-1)

# sequential
def factorialSequential(n):
    if n == 0:
        return 1
    res = 1
    for k in xrange(2, n+1):
        res *= k
    return res
```

The first thing to do is to verify that the three implementations actually give the same results, for this we simply loop over the different functions and try for different values of n. The result is shown in table 3.5.

```
from matrix2latex import matrix2latex
N = range(0, 10)
table = list()
for func in (factorialMath,
             factorialRecursive,
             factorialSequential):
    row = list()
    for n in N:
        res = func(n)  # call func
        row.append(res)# append result to row
    table.append(row)  # append row to table

# row labels
rl = ['$n$', 'Built-in', 'Recursive', 'Sequential']
caption = '''Vertifying that the different factorial
implementations gives the same results'''
matrix2latex(table, 'facV', caption=caption,
             headerColumn=N, headerRow=rl,
             alignment='r', transpose=True)
```

Table 3.5: Vertifying that the different factorial implementations gives the same results

n	Built-in	Recursive	Sequential
0	1	1	1
1	1	1	1
2	2	2	2
3	6	6	6
4	24	24	24
5	120	120	120
6	720	720	720
7	5040	5040	5040
8	40320	40320	40320
9	362880	362880	362880

What we really wanted to do was to compare the speed of the different implementations. To do this we use the python package timeit, shown bellow. The speed comparision is given in table 3.6.

```
import timeit
table = list()
for func in ('factorialMath',
             'factorialRecursive',
             'factorialSequential'):
    row = list()
    for n in N:
        statement = 'factorial.{func}({n})'.format(func=func,
                                                   n=n)
        setup = 'import factorial'
        # measure time
        res = timeit.repeat(statement, setup)
        row.append(min(res)) # append result
    table.append(row) # append row to table

rl = ['$n$', 'Built-in [$s$]',
      'Recursive [$s$]', 'Sequential [$s$]']
caption = '''Comparing execution time for
the different factorial implementations'''
matrix2latex(table, 'facT', caption=caption,
             headerColumn=N, headerRow=rl,
             transpose=True, format='$%.3f$')
```

Table 3.6: Comparing execution time for the different factorial implementations

n	Built-in $[s]$	Recursive $[s]$	Sequential $[s]$
0	0.154	0.212	0.216
1	0.217	0.235	0.639
2	0.213	0.471	0.741
3	0.252	0.685	0.800
4	0.454	0.891	0.873
5	0.486	1.111	0.933
6	0.266	1.319	1.000
7	0.272	1.532	1.063
8	0.283	1.750	1.123
9	0.287	1.977	1.190

As an additional example, see `test_compatibility.py` to see how table

1.2 was created.