Latex output from matlab-octave source blocks

cissic

<2024-04-25 Thu>

Contents

Here are my loose thoughts on the topic started here.

1.1 Source code blocks

If you have Symbolic Matlab Toolbox and some newer version of Matlab you can always use its `latex` function to deal with the problem. For example

```
#+begin_src matlab :session *MatOct* :exports none :eval no-export
  A = [1 2; 2.5 pi] ;
  C = {1 2; "string", { 3.987987933 \pi}} ;
  a = 1 ;
  v = [1 2 3] ;
#+end_src

#+begin_src matlab :session *MatOct* :results output latex :exports results :wrap latex
  disp(['$' latex(sym(A)) '$'])
  disp(['$' latex(sym(C)) '$'])
#+end_src
```

Returns the following results:

$$\begin{pmatrix} 1 & 2 \\ \frac{5}{2} & \pi \end{pmatrix} \begin{pmatrix} 1 & 2 \\ \text{string} & \frac{886943224362615}{1125899906842624} \end{pmatrix}$$

1.2 Inline source blocks

For inline blocks you need to however use `raw` modifier like below:

`src_matlab[:session *MatOct* :results raw]{disp(['$' latex(sym(A)) '$'])}`

which results in:

$\begin{pmatrix} 1 & 2 \\ \frac{5}{2} & \pi \end{pmatrix}$.

One of the pros of `latex(sym(....` approach is that we can easily deal not only with matrices but also with cell arrays. The con is that it converts floats to decimal fractions, which is not always desirable. However...

1.3 Making matlab code blocks be easily converted to octave code blocks

... from my point of view, it would be very convenient to have the ability to convert matlab source blocks easily into octave source blocks and vice versa (when you don't use sophisticated functions from Matlab toolboxes it's better to evoke octave). For such use-cases it'd be good to have consistent way of working with source blocks. Unfortunately I cannot not find it.

First, we cannot use `latex` function from Matlab's Symbolic Toolbox anymore.

OK. We can deal with that by tailoring useful `matrix2latex` function from Mathworks FileExchange. The original version can be found here: matrix2latex. It's drawback is that it stores results in a file which name is given as a function second parameter.

I have edited it a bit to make it return latex code as a string. I have no patience to do it properly, so I only commented out the parts of the original file that refer to saving to the file on the disk.

Here's my version:

%% %%%%%%%%%%%%%%%

```
1  function [S] = matrix2latexS(matrix, filename_NOTUSED, varargin)
2
3  % function: matrix2latex(...)
4  % Author:   M. Koehler
5  % Contact:  koehler@in.tum.de
6  % Version:  1.1
7  % Date:     May 09, 2004
8
9  % This software is published under the GNU GPL, by the free software
```

```
% foundation. For further reading see:
↪  http://www.gnu.org/licenses/licenses.html#GPL

% Usage:
% matrix2late(matrix, filename, varargs)
% where
%   - matrix is a 2 dimensional numerical or cell array
%   - filename is a valid filename, in which the resulting latex code will
%   be stored
%   - varargs is one ore more of the following (denominator, value)
↪  combinations
%      + 'rowLabels', array -> Can be used to label the rows of the
%      resulting latex table
%      + 'columnLabels', array -> Can be used to label the columns of the
%      resulting latex table
%      + 'alignment', 'value' -> Can be used to specify the alginment of
%      the table within the latex document. Valid arguments are: 'l', 'c',
%      and 'r' for left, center, and right, respectively
%      + 'format', 'value' -> Can be used to format the input data.
↪  'value'
%      has to be a valid format string, similar to the ones used in
%      fprintf('format', value);
%      + 'size', 'value' -> One of latex' recognized font-sizes, e.g.
↪  tiny,
%      HUGE, Large, large, LARGE, etc.
%
% Example input:
%   matrix = [1.5 1.764; 3.523 0.2];
%   rowLabels = {'row 1', 'row 2'};
%   columnLabels = {'col 1', 'col 2'};
%   matrix2latex(matrix, 'out.tex', 'rowLabels', rowLabels,
↪  'columnLabels', columnLabels, 'alignment', 'c', 'format', '%-6.2f',
↪  'size', 'tiny');
%
% The resulting latex file can be included into any latex document by:
% /input{out.tex}
%
% Enjoy life!!!

    rowLabels = [];
    colLabels = [];
    alignment = 'l';
    format = [];
    textsize = [];
    if (rem(nargin,2) == 1 || nargin < 2)
        error('matrix2latexS: ', 'Incorrect number of arguments to %s.',
        ↪  mfilename);
    end
```

```
51
52      okargs = {'rowlabels','columnlabels', 'alignment', 'format', 'size'};
53      for j=1:2:(nargin-2)
54          pname = varargin{j};
55          pval = varargin{j+1};
56          k = strmatch(lower(pname), okargs);
57          if isempty(k)
58              error('matrix2latexS: ', 'Unknown parameter name: %s.',
                ↪  pname);
59          elseif length(k)>1
60              error('matrix2latexS: ', 'Ambiguous parameter name: %s.',
                ↪  pname);
61          else
62              switch(k)
63                  case 1  % rowlabels
64                      rowLabels = pval;
65                      if isnumeric(rowLabels)
66                          rowLabels = cellstr(num2str(rowLabels(:)));
67                      end
68                  case 2  % column labels
69                      colLabels = pval;
70                      if isnumeric(colLabels)
71                          colLabels = cellstr(num2str(colLabels(:)));
72                      end
73                  case 3  % alignment
74                      alignment = lower(pval);
75                      if alignment == 'right'
76                          alignment = 'r';
77                      end
78                      if alignment == 'left'
79                          alignment = 'l';
80                      end
81                      if alignment == 'center'
82                          alignment = 'c';
83                      end
84                      if alignment ~= 'l' && alignment ~= 'c' && alignment
                        ↪  ~= 'r'
85                          alignment = 'l';
86                          warning('matrix2latexS: ', 'Unkown alignment. (Set
                            ↪  it to \''left\''.)');
87                      end
88                  case 4  % format
89                      format = lower(pval);
90                  case 5  % format
91                      textsize = pval;
92              end
93          end
94      end
```

```
95
96      S = [''] ; %fid = fopen(filename, 'w');
97
98      width = size(matrix, 2);
99      height = size(matrix, 1);
100
101     if isnumeric(matrix)
102         matrix = num2cell(matrix);
103         for h=1:height
104             for w=1:width
105                 if(~isempty(format))
106                     matrix{h, w} = num2str(matrix{h, w}, format);
107                 else
108                     matrix{h, w} = num2str(matrix{h, w});
109                 end
110             end
111         end
112     end
113
114     if(~isempty(textsize))
115         S = [S sprintf('\\begin{%s}', textsize) ] ; % fprintf(fid,
            ↪  '\\begin{%s}', textsize);
116     end
117
118     S = [S sprintf('\\begin{tabular}{|') ] ;  %fprintf(fid,
        ↪  '\\begin{tabular}{|');
119
120     if(~isempty(rowLabels))
121         S = [S sprintf('l|')] ; % fprintf(fid, 'l|');
122     end
123     for i=1:width
124         S = [S sprintf('%c|', alignment)]; % fprintf(fid, '%c|',
            ↪  alignment);
125     end
126     S = [S sprintf('}')]; % fprintf(fid, '}\r\n');
127
128     S = [S sprintf('\\hline')]; % fprintf(fid, '\\hline\r\n');
129
130     if(~isempty(colLabels))
131         if(~isempty(rowLabels))
132             S = [S sprintf(' & ')]; % fprintf(fid, '&');
133         end
134         for w=1:width-1
135             S = [S sprintf('\\textbf{%s} & ', colLabels{w})]; %
                ↪  fprintf(fid, '\\textbf{%s}&', colLabels{w});
136         end
137         S = [S sprintf('\\textbf{%s}\\\\ \\hline', colLabels{width})]; %
            ↪  fprintf(fid, '\\textbf{%s}\\\\\\hline\r\n', colLabels{width});
```

```
138     end
139
140     for h=1:height
141         if(~isempty(rowLabels))
142             S = [S sprintf('\\textbf{%s} & ', rowLabels{h})]; %
                ↪  fprintf(fid, '\\textbf{%s}&', rowLabels{h});
143         end
144         for w=1:width-1
145             S = [S sprintf('%s & ', matrix{h, w})]; % fprintf(fid, '%s&',
                ↪  matrix{h, w});
146         end
147         S = [S sprintf('%s\\\\ \\hline', matrix{h, width})]; %
            ↪  fprintf(fid, '%s\\\\\\hline\r\n', matrix{h, width});
148     end
149
150     S = [S sprintf('\\end{tabular}')]; % fprintf(fid,
        ↪  '\\end{tabular}\r\n');
151
152     if(~isempty(textsize))
153         S = [S sprintf('\\end{%s}', textsize)]; % fprintf(fid,
            ↪  '\\end{%s}', textsize);
154     end
155
156     % fclose(fid);
```

%% %%%%%%%%%%%%%%

Now, with the use of this function we can generate latex matrix code for the given matlab/octave matrix. However, as the examples below indicate, there are still inconsistencies between matlab and octave source block modifiers. The same modifier value that work well with matlab, return unwanted output in octave and vice versa....

1.3.1 Matlab

1. Source blocks

   For:

   ```
   #+begin_src matlab :session *MatOct* :exports none
     A = [1 2; 2.5 pi] ;
     C = {1 2; "string", { 3.987987933 \pi}} ;
     a = 1 ;
     v = [1 2 3] ;
   #+end_src

   #+begin_src matlab :session *MatOct* :results output :exports results :eval never-
   ```

```
  str = matrix2latexS(A, 'THIS_STRING_IS_NOT_USED', 'alignment', 'c', 'format', '%
  disp(str)
#+end_src
```

We obtain:

1.0000	2.0000
2.5000	3.1416

2. Inline blocks

   `src_matlab[:session *MatOct* :results raw]{disp(['$' str '$'])}`

   returns:

1.0000	2.0000
2.5000	3.1416

   Of course the form of latex matrices can be tailored to your needs by adjusting `matrix2latexS` function.

1.3.2 Octave

... and this is where I fall... As I said, it is important to me to have easily convertible matlab code blocks to octave code blocks. However I am not able to find any common, consistent way of accessing results of code blocks of these two languages. Have a look at the examples below:

1. Source blocks

```
#+begin_src octave :session *OctMat*
  A = [1 2; 2.5 pi] ;
  C = {1 2; "string", { 3.987987933 \pi}} ;
  a = 1 ;
  v = [1 2 3] ;
#+end_src

#+begin_src octave :session *OctMat* :results output :exports results :eval never-
  str = matrix2latexS(A, 'THIS_STRING_IS_NOT_USED', 'alignment', 'c', 'format', '%
  ans = str
#+end_src
```

```
A = [1 2; 2.5 pi] ;
C = {1 2; "string", { 3.987987933 \pi}} ;
a = 1 ;
v = [1 2 3] ;
```

   Results in:

```
#+RESULTS:
#+begin_latex
| octave> octave> ans = \begin{tiny}\begin{tabular}{ | c | c | }\hline1.0000 & 2.0
#+end_latex
```

which renders as:

octave> octave> ans = $\begin{smallmatrix}\text{c}\\ \text{c}\end{smallmatrix}$ }height1.0000 & 2.0000\\ height2.5000 & 3.1416\\ height

2. Inline blocks:

   On the other hand this piece of code:

   ```
   src_octave[:session *OctMat* :results raw]{disp(['$' 'string'
   '$'])}
   ```

   results in:

   org$_{\text{babeleoe}}$

1.4 Summary

Org-babel with octave/matlab is a tricky machinery to me.

I'm not sure if it is possible to get both matlab and octave code blocks (and inline blocks) working in the same manner.

If it was, I could adjust `matrix2latex` function to have common way of working with both languages.