The MAT2DOC documentation system

Peter L. Søndergaard

July 17, 2014

1 About mat2doc

MAT2DOC is a system made for extracting documenation from the Matlab function headers. You write the documentation in a Wiki-like syntax, and the system produces output for three different output formats:

- Plain text with simple rendered formulas and no images
- HTML / php with javascript formulas and images
- Latex for processing with pdflatex
- Git or Subversion for managing the source code

The software is written in Python as a wrapper around reStructuredText, and it can therefore handle all the markup possible in reStructuredText, with a few extra shortcuts suitable for writing documenation.The easiest way of getting started is to look at the source code for LTFAT or AMTOOLBOX for a lot of examples.

The restructured-text markup language is explained with a lot of examples on the homepage of the project http://docutils.sourceforge.net/rst.html.

Cyrrently, the system runs only on Linux. No official release has been made yet, but we are working on it.

2 The markup language

Read this section to learn about how to write the headers of function files and demos.

2.1 Code and variables

Code and variables are typeset by changing the font

- Code in always surrounded by backticks as in ‘c=dgt(...)‘ and is typeset in a typewriter font: `c=dgt(...)` The backtics is simply the “default role” in reStructuredText that has been assigned to this.
- Single variables appearing in the text should be surrounded by stars as in *siglen* and is typeset in italics *siglen*
- If you have code in a separate environment spanning one or more lines, close the previous environment by two colons :: . This is standard reStructruredText, so see the documenation for examples.
- If you have example code that should be executed, do as above but end with three (3) colons ::: . You can have multiple example blocks this way, but they are evaluated independently, so variables etc. cannot be shared. You can generate plot in the code, they will captured as images and out into the documenation.

2.2 Math

Formulas are typeset using MathJax which is an open source javascript system, http://www.mathjax.org/, that converts Latex mathematics markup into MathML. MAT2DOC supports typesetting either single inline formulas or display math:

- Simple inline formulas l^2 or $M\times N$ becomes l^2 or $M \times N$. This is a simple extension of reStructuredText, so that you don't have to write :math:`l^2` all the time, but can use the familiar TEX construct.
- Display math formulas are typeset as a separate environment, opening with `.. math::` This is standard reStructuredText.

The inline formulas must be *very* simple, as it must be able to convert them to a simple ascii representation for the plain text output.

The display math formulas can contain any standard latex math markup, but they will not appear in the plain text output. Instead it is possible to create a restructured text comment starting by two full stops and a space. The comment will appear in the plain text output, but nowhere else, so you can manually render the formula in ascii.

2.3 Links

Links to other functions are written between pipe symbols with a final underscore: `See the help on |dgt|_`. This is a standard restructured text substitution, but MAT2DOC will generate the correct substitutions. Note that you can only link to functions in your own toolbox in this way, as MAT2DOC would not know where to link. It is possible to supply a list of substitutions, see Section XXX configuration.

Normal HTML links are written as: `<http://dsp.rice.edu/software/bat-echolocation-chirp>`_

2.4 Itemize and enumerate

You can create itemize and enumerate environments:

- * gives you an "itemize" section
- 1) gives you an "enumerate" section

Environments can be nested by adding spaces in front. Note the correct spacing:

```
1) This is an enumerate item spanning severeal line. Note
   that the second line must be indented relative to the first
   one, otherwise you will get an error.

* Itemize paragraphs simply start with a single star, and
  continue in the same way as the enumeration paragraphs.
```

2.5 Headlines

Headlines are created by underlining:

```
Headline 1:
-----------
```

It is possible to create many types of headlines by using different types of underlining, but it is recommended to stick to only using —-, as one level of headlines should suffice for simple documentation. MAT2DOC will automatically generate some headlines using —-, so if you use another characther you will break the system.

2.6 Lists

Usage lists are written as follows:

```
%   Usage:  c=dgt(f,g,a,M);
%           c=dgt(f,g,a,M,L);
%           c=dgt(f,g,a,M,'lt',lt);
```

they must follow straight after the function definition.

Input parameter lists always start with the *exact* headline: "Input parameters:" as in

```
%   Input parameters:
%         f     : Input data.
%         g     : Window function.
```

Similarly for output parameters

```
%   Output parameters:
%         c     : $M \times N$ array of coefficients.
%         Ls    : Length of input signal.
```

The right hand side of the parameter definitions are full reStructuredText blocks, so you can put in markup code and empty lines to create blocks.

There is special support for optional parameter list in the ltfatarghelper style, see Section 4. They are written like this:

```
%     'dynrange',r  This is a key/value pair.
%
%     'db'          This is a flag definition. The description may
%                   span multiple lines, and may include markup like $l^2$.
```

Some important details:

- There are 5 spaces between the percent sign and starting description of the parameters
- Each parameter starts with an apostrophe '
- For key/value pairs, you can add the variable after the flag
- There *must* be at least two spaces between the parameter name and the start of the description (so in the above example between "'dynrange',r" and "This is a key ...").

2.7 Stuff at the end

You can add references to the end of you function by writing the key to the reference as in the following:

```
%   References:  griffin1984sem fest98
```

This will expand the citations as text in the Matlab function file, as HTML code in the HTML output and as a regular citation in the Latex output. The keys are looked up in the bibtex file of you project, see the configuration section XXX.

If a line opens by see also:, the "see also" items will be converted into links to the relevant functions:

```
% See also: dgt, dgtreal
```

If you have many see also items, it is possible for them to span multiple lines as in:

```
See also: takanen2013periphery, takanen2013mso,
          takanen2013lso, takanen2013wbmso,
          takanen2013directionmapping
```

Complete similar to "See also" it is possible to specify demos by a line opening with the word "Demos:"

2.8 A complete example

This section describes the overall structure and gives a complete example of a function. It is a reduced version of the dgt

```
01 function [c,Ls,g]=dgt(f,g,a,M,varargin)
02 %DGT  Discrete Gabor transform
03 %   Usage:  c=dgt(f,g,a,M);
04 %           c=dgt(f,g,a,M,L);
06 %
07 %   Input parameters:
08 %         f     : Input data.
09 %         g     : Window function.
10 %   Output parameters:
11 %         c     : $M \times N$ array of coefficients.
12 %         Ls    : Length of input signal.
13 %
14 %   `dgt(f,g,a,M)` computes the Gabor coefficients of the input signal
15 %   *f* with respect to the window *g* and parameters *a* and *M*. The
16 %   output is a vector/matrix in a rectangular layout.
17 %
18 %   If *f* is a matrix, the transformation is applied to each
19 %   column. The length of the transform done can be obtained by
20 %   `L=size(c,2)*a;`
21 %
22 %   The window *g* may be a vector of numerical values, a text string
23 %   or a cell array. See the help of |gabwin|_ for more details.
24 %
25 %   `dgt(f,g,a,M,L)` computes the Gabor coefficients as above, but does
26 %   a transform of length *L*. f will be cut or zero-extended to length
27 %   *L* before the transform is done.
28 %
29 %   `[c,Ls]=dgt(f,g,a,M)` or `[c,Ls]=dgt(f,g,a,M,L)` additionally
30 %   returns the length of the input signal *f*. This is handy for
31 %   reconstruction::
32 %
33 %               [c,Ls]=dgt(f,g,a,M);
34 %               fr=idgt(c,gd,a,Ls);
35 %
36 %   will reconstruct the signal *f* no matter what the length of *f* is,
37 %   provided that *gd* is a dual window of *g*.
38 %
39 %   The Discrete Gabor Transform is defined as follows: Consider a
40 %   window *g* and a one-dimensional signal *f* of length *L* and
41 %   define $N=L/a$. The output from `c=dgt(f,g,a,M)` is then given by:
42 %
43 %   ..              L-1
44 %      c(m+1,n+1) = sum f(l+1)*conj(g(l-a*n+1))*exp(-2*pi*i*m*l/M),
45 %                   l=0
46 %
47 %   .. math:: c\left(m+1,n+1\right) =
48 %      \sum_{l=0}^{L-1}f(l+1)\overline{g(l-an+1)}e^{-2\pi ilm/M}
49 %
50 %   where $m=0,\ldots,M-1$ and $n=0,\ldots,N-1$ and $l-an$ is computed
51 %   modulo *L*.
52 %
53 %   Additional parameters:
```

```
54 %   ----------------------
55 %
56 %   `dgt` takes the following flags at the end of the line of input
57 %   arguments:
58 %
59 %     'freqinv'  Compute a DGT using a frequency-invariant phase. This
60 %                is the default convention described above.
61 %
62 %     'timeinv'  Compute a DGT using a time-invariant phase. This
63 %                convention is typically used in filter bank algorithms.
64 %
65 %   Examples:
66 %   ---------
67 %
68 %   In the following example we create a Hermite function, which is a
69 %   complex-valued function with a circular spectrogram, and visualize
70 %   the coefficients using both `imagesc` and |plotdgt|_:::
71 %
72 %     a=10;
73 %     M=40;
74 %     L=a*M;
75 %     h=pherm(L,4); % 4th order hermite function.
76 %     c=dgt(h,'gauss',a,M);
77 %
78 %     % Simple plot: The squared modulus of the coefficients on
79 %     % a linear scale
80 %     figure(1);
81 %     imagesc(abs(c).^2);
82 %
83 %   See also:  idgt, gabwin, dwilt, gabdual, phaselock
84 %
85 %   Demos:  demo_dgt
86 %
87 %   References: fest98 gr01
```

A description of the lines:

01 This is the function header, it must always be the first line, otherwise the function will be treated as an executable demo

02 The header of the help section. It must start with the name of the function in capital letters followed by a one-line description of the function. The description must not be too long, as it will be used for headlines in the html and tex output.

03-04 The usage section

07-12 Input and output parameters

14-16 The first real description of the function, telling the user what the function does for its most common set of parameters. Each parameter is desribed shortly. The function and its parameters appear first on the line surrounded by backticks. Notice that the free variables are surrounded by stars.

18-20 Another paragraph in the description. Notice the inline code on line 20

23 This is a link to another function, it will be uppercase in Matlab and links will be generated in the other targets.

25-31 These paragraphs describes less common ways of calling the function by referring to the previous paragraph.

33-34 This is inactive code, it is places here to explain something to the user, but the code itself cannot be executed as such. Notice how the previous paragraph is closed by two colons in line 31 to mark the code block.

43-45 This is a reStructuredTex comment. The text one these lines will *only* appear in the Matlab output. This is done here to manually render the formula below.

47-48 This is the definition of a display math formula. It will appear in the HTML and tex output.

53-54 This is a headline, notice the correct underlining.

59-63 This is description of optional input parameters. Notice the 5 spaces before the parameters and the 2 spaces in between the label and the description.

72-81 This is an active code block. Notice the three colons finishing line 70 to mark the beginning. This code block will generate a figure that will be captured for the HTML and tex output.

83 The "See also" line

85 The "Demos" line

87 The "References" line. This is the recommended ordering of the three lines.

2.9 demos

A demo is simple a script file, not starting with the "function" command. Demos are executed by mat2doc, and the figures are output are saved, just as for examples in the help section. The major difference is that captions for generated images are written as in the following example:

```
01 %DEMO_AUDSCALES  Plot of the different auditory scales
02 %
03 %   This demos generates a simple figure that shows the behaviour of
04 %   the different audiory scales in the frequency range from 0 to 8000 Hz.
05 %
06 %   .. figure::
07 %
08 %      Auditory scales
09 %
10 %      The figure shows the behaviour of the audiory scales on a normalized
11 %      frequency plot.
12 %
13 %   See also:  freqtoaud, audtofreq, audspace, audspacebw
```

The opening line (01) is completely similar to the opening of a function, but there are no "Usage", "Input Parameters" or "Output Parameters" sections. Lines 06-11 describes the first figure generated by the demo. The structure for a figure is always as follows:

1. The line ".. figure::" (line 06) marking the beginning of the figure definition. It must be followed by an empty line
2. A *single* line (line 08) providing the caption for the figure. It must be followed by an empty line
3. Regular text, perhaps including several paragraphs that describe the figure.

This is a simple wrapper around the restructuredtext "Figure " directive: http://docutils.sourceforge.net/docs/ref/rst/d
MAT2DOC simply inserts the correct file name and properties after the first line.

3 Setting up mat2doc

Read this section if you want to set up MAT2DOC for your own project.

3.1 Installation

You will need the following programs

- Python 2.7 with the following modules: docutils and pygments
- Lynx
- Bibtex2html
- Octave or Matlab

3.2 Configuration

Configuration is done inside a mat2doc directory in you package, so if you package is stored in `/path/to/mypackage`, then all the cofiguration is stored in `/path/to/mypackage/mat2doc`

The main configuration file is called "conf.py". This is parsed as a regular Python file, and you can set variables for your whole project. The current supported variables are:

3.2.1 Mandatory parameters

`outputdir` This is the directory where the output mat/php/tex output will be stored.

`plotengine` Which program to use for plotting, values are either `'matlab'` or `'octave'`.

`version` Version number of your package as a string.

`versionfile` Name of a file that contains the version number as a string. You must specify either `'version'` or `'versionfile'`.

3.2.2 Optional parameters

`author` Name of the author.

`year` The year (as a string)

`octtitle` The first line of text for the Octave package INDEX file

`matlabexec` Full path and name of the Matlab executable

`lynxexec` Full path and name of the Matlab executable

`lynxdir` Directory containing the lynx.cfg configuration file

`octaveexec` Full path and name of the Matlab executable

`bibtex2htmlexec` Full path and name of the Matlab executable

`gitexec` Full path and name of the Git executable

`svnexec` Full path and name of the Subversion executable

3.3 Running mat2doc

mat2doc is a executable Python file, so on Linux/Unix you can set the executable permissions and run it directly, as in

```
mat2doc.py /path/to/mypackage mat
```

but on Windows you will need to first call python as in

```
python mat2doc.py /path/to/mypackage mat
```

The first argument of mat2doc is the full directory path to you package (you can use ~ on Linux).

The second argument is the *target* to generate:

`mat` Generate the .m files. This creates a directory with the files that the user should download

`php` Generate php files

`html` Generate plain html files

`tex` Generate Latex files

The output will be placed in a subdirectory of the output directory specified in the main `conf.py` file, so the "mat" output can be found in `/path/to/outputdir/packagename-mat` and similarly for the other targets.

4 The ltfatarghelper function

Many functions needs optional parameters, and Matlab does not have a decisive system for specifying default values. `ltfatarghelper` is a homegrown system for parsing `varargin`. It is simply one function bundled with LTFAT. It supports:

- Key / value pair
- flags organized in groups
- Positional key / value pairs
- Expansion of a single flag into a default parameter line
- Inheriting definitions from other functions

Here is a simple example on how to use `ltfatarghelper`.In the header of the function that uses `ltfatarghelper`, put the `varargin` at the end:

```
function [c,Ls,g]=dgt(f,g,a,M,varargin)
```

In the function, before doing anything else, put the following:

```
definput.keyvals.L=[];
definput.keyvals.lt=[0 1];
definput.flags.phase={'freqinv','timeinv'};
[flags,kv,L]=ltfatarghelper({'L'},definput,varargin);
```

This defines

1. A key/value pair L with having the empty matrix as the default value.
2. A key/value pair lt having a 1×2 matrix as the default value
3. Two mutually exclusive flags 'freqinv' and 'timeinv'. If no flag is specified, the first one is assumed, in this case 'freqinv'

The last line call `ltfatarghelper` to parse the input line with the additional specification that L may be entered first in the line of optional parameters as an optional parameter.

These are some of the possibilities that the user can write

- `dgt(f,g,a,M,L)`
- `dgt(f,g,a,M,L,'timeinv')`
- `dgt(f,g,a,M,'timeinv')`
- `dgt(f,g,a,M,'lt',[1 2])`

After the call you can refer to these variables in your code

- `L`
- `kv.L`
- `kv.lt`
- `flags.do_freqinv`
- `flags.do_timeinv`