Pweave - literate programming with Python

Release 0.21.1

Matti Pastell

May 16, 2013

Contents

Contents:

1 Pweave Basics

1.1 Pweave syntax

Pweave uses noweb syntax for defining the code chunks and documentation chunks, just like Sweave.

Code chunk
: start with a line marked with <<>>= or <<options>>= and end with line marked with @. The code between the start and end markers is executed and the output is captured to the output document. See for options below.

  Example: A code chunk that saves and displays a 12 cm wide image and hides the source code:

```
<<fig = True, width = '12 cm', echo = False>>=
from pylab import *
plot(arange(10))
show()
@
```

Documentation chunk
: Are the rest of the document (between @ and <<>>= lines and the first chunk be default) and can be written with several different markup languages.

Inline code
: Pweave supports evaluating inline code in documentation chunks using `<% %>` (code will be evaluated in place) and `<%= %>` (result of expression will be printed) tags. Inline code will not be included in weaved document.

  Example: Use inline code to set some matplotlib options.

```
<%
import matplotlib
matplotlib.rcParams.update({'figure.figsize' : (6, 4),
                            'savefig.dpi': 100,
                            'font.size' : 10 })
%>
```

New in version 0.2.

1.2 Weaving Pweave Documents

Weaving a Pweave source file produces a document that contains text and the weaved code together with its evaluated output. All of the produced figures are placed in the 'figures/' folder as a default.

See formats page for a list of supported output formats.

Pweave documents are weaved from the shell with the command:

`Pweave [options] sourcefile`

Options:

`-version`
: show the version number and exit

`-h, -help`
: show help message and exit

`-f` `FORMAT`, `-format` `FORMAT`
: The output format. Available formats: sphinx, pandoc, tex, html, rst, texpweave, texminted. See http://mpastell.com/pweave/formats.html

`-m` `MPLOTLIB`, `-matplotlib`=`MPLOTLIB`
: Do you want to use matplotlib (or Sho with Ironpython) true (default) or false

`-d, -documentation-mode`
: Use documentation mode, chunk code and results will be loaded from cache and inline code will be hidden

`-c, -cache-results`
: Cache results to disk for documentation mode

`-figure-directory`=`FIGDIR`
: Directory path for matplolib graphics: Default 'figures'

`-cache-directory`=`CACHEDIR`
: Directory path for cached results used in documentation mode: Default 'cache'

-g FIGFMT, **-figure-format**=FIGFMT
: Figure format for matplolib graphics: Defaults to 'png' for rst and Sphinx html documents and 'pdf' for tex

Weave a document with default options (rst with png figures)

```
$ Pweave ma.Pnw
Output written to ma.rst
```

Weave a Latex document with png figures:

```
$ Pweave -f tex -g png source.Pnw
```

Get options:

```
$ Pweave --help
```

1.3 Tangling Pweave Documents

Tangling refers to extracting the source code from Pweave document. This can be done using Ptangle script:

```
$ Ptangle file

$ Ptangle ma.pnw
Tangled code from ma.pnw to ma.py
```

1.4 Code Chunk Options

Pweave currently has the following options for processing the code chunks.

name, label
: If the first option of chunk is unnamed it will become the chunk name, you can also set the chunk name using the name or label (*for Sweave compatibility*) keys. All of these definitions are equal `<<analysis, Fig = True>>=`, `<<Fig = True, name = 'analysis'>>=`, `<<Fig = True, label = 'analysis'>>=`. Chunk names are used for figure names, but expanding named chunks in the Pweave todo list.

New in version 0.2.

fig = True or (False)
: Whether a matplotlib plot produced by the code chunk should be included in the file. The figure will be added with '.. image::' directive in .rst and \includegraphics tag in .tex documents. See the 'caption' option if you want to use figure environment. As of version 0.21 Pweave supports multiple figures per code chunk.

include = True or (False)

: **If include is True generated figures are automatically included in** the document otherwise figures are generated, but not included. This is useful if you want more control over figure formatting e.g. use subfigures in Latex.

New in version 0.21.

width
: The width of the created figure (using format specific markup). The default width depends on the otuput format.

echo = True or (False)
: Echo the python code in the output document. If False the source code will be hidden.

evaluate = True or (False).
: Evaluate the code chunk. If False the chunk won't be executed.

results = 'verbatim'
: The output format of the printed results. 'verbatim' for literal block, 'hidden' for hidden results or anything other string for raw output (I tend to use 'tex' for Latex and 'rst' for rest. Raw output is useful if you wan't to e.g. create tables from code chunks.

caption = ' `
: A string providing a caption for the figure produced in the code chunk. Can only be used with 'fig = True' option. If a caption is provided the figure will be added in the .rst document with the '.. figure::' directive and as a figure float in Latex.

New in version 0.12.

term = False or (True)
: If True the output emulates a terminal session i.e. the code chunk and the output will be printed as a doctest block. Can also be used in latex documents, where the output will formatted as verbatim.

f_pos = "htpb"
: Sets the figure position for latex figures.

New in version 0.21.

wrap = True or (False,"code", "results")
: Controls wrapping of long lines. If True both code and output are wrapped to 75 characters. You can also specify "code" or "results" options to wrap only input or output.

New in version 0.21.

Note: You can now use loops, if sentences and function definitions in term chunks (as of Pweave 0.13). However sometimes execution in term mode can fail in which case the chunk will be executed with term=False.

New in version 0.12.

1.5 Changing defaults

Default chunk options are stored in *pweave.Pweb.defaultoptions* dictionary. You can manipulate the dictionary to change the options.

Have a look at current defaults:

```
from pweave import *
import pprint
pprint.pprint(Pweb.defaultoptions)
```

```
{'caption': False,
 'complete': True,
 'echo': True,
 'evaluate': True,
 'f_env': None,
 'f_pos': 'htpb',
 'f_size': (8, 6),
 'fig': True,
 'include': True,
 'name': None,
 'results': 'verbatim',
 'term': False,
 'wrap': True}
```

Change wrapping off and default figure position to "h!"

```
Pweb.defaultoptions.update({'wrap' : False, 'f_pos' : "h!"})
#Updated options
pprint.pprint(Pweb.defaultoptions)
```

```
{'caption': False,
 'complete': True,
 'echo': True,
 'evaluate': True,
 'f_env': None,
 'f_pos': 'h!',
 'f_size': (8, 6),
 'fig': True,
 'include': True,
 'name': None,
 'results': 'verbatim',
 'term': False,
 'wrap': False}
```

1.6 Caching results

Pweave has documentation mode (invoked with `-d`) that caches code and all results from code chunks so you don't need to rerun the code when you are only working on documentation. You can cache the results using the *-c* option, if there are no cached results then documentation mode will create the cache on first run. Inline code chunks will be hidden in documentation mode. Additionally Pweave will warn you if the code in cached chunks has changed after the last run.

1.7 Document types

Source document
: Contains a mixture of documentation and code chunks. Pweave will evaluate the code and leave the documentation chunks as they are. The documentation chunks can be written either with reST,Latex or Pandoc markdown. The source document is processed using *Pweave*, which gives us the formatted output document.

Weaved document
: Is produced by Pweave from the source document. Contains the documentation, original code, the captured outputof the code and optionally captured matplotlib figures.

Source code
: Is produced by Pweave from the source document. Contains the source code extracted from the code chunks.

2 Output Formats

2.1 Listing formats

Pweave supports output in several formats. You can list the supported formats using:

```
from pweave import PwebFormats
PwebFormats.listformats()

Pweave supported output formats:

* html:
   HTML with pygments highlighting
* md2html:
   Markdown to HTML using Python-Markdown
* pandoc:
   Pandoc markdown
* pandoc2html:
   Markdown to HTML using Pandoc, requires Pandoc in path
* pandoc2latex:
   Markdown to Latex using Pandoc, requires Pandoc in path
* rst:
   reStructuredText
* sphinx:
   reStructuredText for Sphinx
* tex:
   Latex with verbatim for code and results
* texminted:
   Latex with predefined minted environment for codeblocks
* texpweave:
   Latex output with user defined formatting using named environments (in latex
* texpygments:
   Latex output with pygments highlighted output
```

```
More info: http://mpastell.com/pweave/formats.html
```

2.2 Format descriptions

rst
: reStructuredText . See *reST example*.

tex
: Standard LaTeX. Code, results and terminal blocks are written using *verbatim* environment. Sample document and pdf output.

texminted
: LaTeX with preset minted formatting for code and results. Sample document and pdf output.

texpweave
: LaTeX where code is written using *pweavecode*, results using *pweaveout* and terminals using *pweaveterm* environment. The user can (and needs to) define the formatting for these environment in preamble. This can done e.g. with the *\newminted* command with minted-package.

pandoc
: Pandoc markdown.

sphinx
: reStructuredText for Sphinx . See *Output of the example in Sphinx*.

html
: HTML with pygments highlighting for code. You'll need to add css yourself, here's one option pygments.css . Sample ma.html.

3 Using pweave module

Pweave can also be used as module from the Python interpreter. This has some advantages over just using the scripts. First the execution of the code will be faster because all modules all already imported. Second you can work interactively with the data after the code from the document has been run. Further it is possible to fully customize the document execution and formatting using the `Pweb` class.

pweave module contains two functions `pweave()` and `ptangle()` that offer the same functionality as the command line scripts.

Note: This document was also created with Pweave, have a look at the source.

3.1 Simple weaving and tangling:

Here's and example of simple weaving and tangling using example document ma.Pnw. Notice that pweave prints out the progress so in case of an error you can tell in which chunk it occurred. Also in case of an error returns already evaluated results from the documents namespace `Pweb.globals` to global namespace.

```
>>> import pweave
>>> # Weave a document with default options
>>> pweave.pweave('ma.Pnw')
Processing chunk 1 named None
Processing chunk 2 named None
Processing chunk 3 named None
Processing chunk 4 named None
Pweaved ma.Pnw to ma.rst
>>> # Extract the code
>>> pweave.ptangle('ma.Pnw')
Tangled code from ma.Pnw to ma.py
```

pweave and pweave function

`pweave.`**`pweave`**(*file*, *doctype='rst'*, *informat='noweb'*, *plot=True*, *docmode=False*, *cache=False*, *figdir='figures'*, *cachedir='cache'*, *figformat=None*, *returnglobals=True*, *listformats=False*)

Processes a Pweave document and writes output to a file

Parameters

- **file** – `string` input file
- **doctype** – `string` output document format: call with listformats true to get list of supported formats.
- **informat** – `string` input format: "noweb" or "script"
- **plot** – `bool` use matplotlib (or Sho with Ironpython)
- **docmode** – `bool` use documentation mode, chunk code and results will be loaded from cache and inline code will be hidden
- **cache** – `bool` Cache results to disk for documentation mode
- **figdir** – `string` directory path for figures
- **cachedir** – `string` directory path for cached results used in documentation mode
- **figformat** – `string` format for saved figures (e.g. '.png'), if None then the default for each format is used
- **returnglobals** – `bool` if True the namespace of the executed document is added to callers global dictionary. Then it is possible to work interactively with the data while writing the document. IronPython needs to be started with -X:Frames or this won't work.

- **listformats** – `bool` List available formats and exit

`pweave.`**`ptangle`**(*file*)
Tangles a noweb file i.e. extracts code from code chunks to a .py file

Parameters **file** – `string` the pweave document containing the code

3.2 More options with Pweb Class

Weaving, tangling and pweave options are implemented using `Pweb` class. There is an example about customizations and the class reference is below.

Pweb Class

class `pweave.`**`Pweb`**(*file=None*, *format='tex'*)
Processes a complete document

Parameters

- **file** – `string` name of the input document.
- **format** – `string` output format from supported formats. See: http://mpastell.com/pweave/formats.html

`cachedir` = **'cache'**
Pweave cache directory

`defaultoptions` = **{'term': False, 'complete': True, 'f_pos': 'htpb', 'evaluate': True, 'f_env':**
Default options for chunks

`documentationmode` = **None**
Use documentation mode?

`figdir` = **'figures'**
Pweave figure directory

`format`()
Format the code for writing

`getformat`()
Get current format dictionary. See: http://mpastell.com/pweave/customizing.html

`globals` = **{}**
Globals dictionary used when evaluating code

`parse`(*string=None*, *basename='string_input'*)
Parse document

`run`()
Execute code in the document

`setformat`(*doctype='tex'*, *Formatter=None*)
Set output format for the document

Parameters

- **doctype** – `string` output format from supported formats. See: http://mpastell.com/pweave/formats.html
- **Formatter** – Formatter class, can be used to specify custom formatters. See: http://mpastell.com/pweave/subclassing.html

setreader (*Reader=<class 'pweave.readers.PwebReader'>*)
Set class reading for reading documents, readers can be used to implement different input markups

tangle ()
Tangle the document

updateformat (*dict*)
Update existing format, See: http://mpastell.com/pweave/customizing.html

usematplotlib = True
Use plots?

weave ()
Weave the document, equals -> parse, run, format, write

write (*action='Pweaved'*)
Write formatted code to file

4 Customizing output

Pweave has several output formats and you can customize the output with chunk options. However you may want to customize the output for different purposes.

The simplest form of customization is to update the *format dictionary* of an existing format. It sets chunk delimiters, output extension and figure format and width.

You can do this easily with `Pweb` class. Below is a small demonstration using ReST Pweave document ma2.Pnw.

Let's start by creating an instance of `Pweb` class with rst document:

```
>>> from pweave import *
>>> from pprint import pprint
>>> doc = Pweb('ma2.Pnw', format = "rst")
```

Have a look at what the format dictionary contains:

```
>>> pprint(doc.getformat())
{'codeend': '\n\n',
 'codestart': '.. code:: python\n',
 'doctype': 'rst',
 'extension': 'rst',
 'figfmt': '.png',
 'indent': '    ',
```

```
 'outputend': '\n\n',
 'outputstart': '::\n',
 'savedformats': ['.png'],
 'termend': '\n\n',
 'termindent': '',
 'termstart': '',
 'width': '15 cm'}
```

The names of the dictionary elements are hopefully self explanatory. You'll notice that you can specify start and end tag for code, results and term as well as block indent.

You can change the formats using `Pweb.updateformat()` method. Let's set the default figure width to 10cm and figure format to pdf. The `savedformats` key allows you to specify multiple formats to save and `figfmt` specifies what format is used in the output.

```
>>> doc.updateformat({'width' : '10cm', 'figfmt' : '.pdf', 'savedformats' : ['.p
df']})
```

And after setting options weave and tangle the document:

```
>>> doc.weave()
Processing chunk 1 named None
Processing chunk 2 named None
Processing chunk 3 named None
Processing chunk 4 named None
Pweaved ma2.Pnw to ma2.rst
>>> doc.tangle()
Tangled code from ma2.Pnw to ma2.py
```

View this page as Pweave document .

5 Subclassing formatters

In the previous section we customized the output format by altering the format dictionary. Sometimes more advanced customizations are needed. This can be done by subclassing Existing formatters .

The base class PwebFormatter has a method `preformat_chunk()` that processes all chunks before they are processed by default formatters.

Suppose I have this document (view the source in browser) using markdown markup and I want convert the doc chunks to HTML and output code chunks using Pweave default HTML formatter.

I can do this by subclassing `PwebHTMLFormatter`. MDtoHTML class below converts the content of all documentation chunks to HTML using python Markdown package. (`chunk['type']` for code chunks is "code"). *The class also stores the chunks for us to see what they contain, but that's not needed for formatting.*

```
from pweave import *
import markdown
```

```
class MDtoHTML(PwebHTMLFormatter):

    chunks = [] #Let's keep a copy of chunks

    def preformat_chunk(self, chunk):
        MDtoHTML.chunks.append(chunk.copy()) #Store the chunks
        if chunk['type'] == "doc":
            chunk['content'] = markdown.markdown(chunk['content'])
        return(chunk)
```

The specified subclass can then be used as formatter with `Pweb` class.

```
doc = Pweb('ma.mdw')
doc.setformat(Formatter = MDtoHTML)
doc.weave()
```

```
Processing chunk 1 named None
Processing chunk 2 named None
Processing chunk 3 named None
Pweaved ma.mdw to ma.html
```

And here is the weaved document.

5.1 A closer look at the chunks

Remember that we kept a copy of the chunks in the previous example? As you can see below the chunk is a dictionary that contains code, results and all of the chunk options. You can manipulate all of these options as we did to content in previous example to control how the chunk is formatted in output.

Note: You can your own options (key = value) to chunks and they will also appear in the chunk dictionary.

Let's see what the first code chunk contains:

```
import pprint
pprint.pprint(MDtoHTML.chunks[1])
```

```
{'caption': False,
 'codeend': '',
 'codestart': '',
 'complete': True,
 'content': '\nfrom pylab import *\nimport scipy.signal as signal\n#A function t
o plot frequency and phase response\ndef mfreqz(b,a=1):\n    w,h = signal.freqz(
b,a)\n    h = abs(h)\n    return(w/max(w), h)\n',
 'doctype': 'html',
 'echo': True,
 'evaluate': True,
 'extension': 'html',
```

```
 'f_env': None,
 'f_pos': 'htpb',
 'f_size': (8, 6),
 'fig': True,
 'figfmt': '.png',
 'figure': [],
 'include': True,
 'name': None,
 'number': 1,
 'outputend': '',
 'outputstart': '',
 'result': '\n\n',
 'results': 'verbatim',
 'savedformats': ['.png'],
 'term': False,
 'termend': '',
 'termstart': '',
 'type': 'code',
 'width': '600',
 'wrap': True}
```

Note: Pweb class also uses separate classes to parse and execute the document, but subclassing these is not currently documented and is hopefully not needed.

---

6 reST example

Here is a simple example of a Pweave file (ma.Pnw) that uses reST as the documentation format. The file demonstrates basic usage of Pweave and how it can easily be used to add dynamic figures and tables.

The file was processed with Pweave using:

```
Pweave -f sphinx --figure-directory=_static ma.Pnw
```

And as result we get the reST document ma.rst (shown below) which uses the Sphinx markup. *If you want to get pure reST that can be processed with e.g. rst2html document you need to call Pweave with '-f rst' option.*

```
Pweave Example - Frequency response of a moving average filter

============================================================



:Author: Matti Pastell <matti.pastell@helsinki.fi>

:Website: http://mpastell.com
```

**Create 11 point moving average filter and plot its frequency response and prin

.. code:: python

```
    from pylab import *
    import scipy.signal as signal
    #A function to plot frequency and phase response
    def mfreqz(b,a=1):
        w,h = signal.freqz(b,a)
        h = abs(h)
        return(w/max(w), h)
```

**Make the impulse response function and use terminal formatted output (=doctest

```
>>> n = 11.
>>> n
11.0
>>> b = repeat(1/n, n)
>>> b
```

```
array([ 0.09090909,  0.09090909,  0.09090909,  0.09090909,  0.09090909,
        0.09090909,  0.09090909,  0.09090909,  0.09090909,  0.09090909,
        0.09090909])
```

Calculate the frequency response and plot it:

```
.. code:: python

    w, h = mfreqz(b)
    #Plot the function
    plot(w,h,'k')
    ylabel('Amplitude')
    xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
    show()
```

```
.. figure:: figures/ma_figure3_1.png
   :width: 15 cm

   Frequency response of an 11 point moving average filter
```

```
**The first 10 values of the frequency response (w,h) as a table, notice that th


.. csv-table::

   :header: "Amplitude", "Frequency"

   :widths: 15, 15




   1.0 , 0.0

   1.0 , 0.0

   1.0 , 0.0

   1.0 , 0.01

   1.0 , 0.01

   1.0 , 0.01

   0.99 , 0.01

   0.99 , 0.01

   0.99 , 0.02

   0.98 , 0.02
```

6.1 Output of the example in Sphinx

The produced rst file be included in a Sphinx document, like this website using ”.. include:: ma.rst”. Here is what it looks like:

Pweave Example - Frequency response of a moving average filter

Author Matti Pastell <matti.pastell@helsinki.fi>

Website http://mpastell.com

Create 11 point moving average filter and plot its frequency response and print the values.

```
from pylab import *
import scipy.signal as signal
#A function to plot frequency and phase response
def mfreqz(b,a=1):
    w,h = signal.freqz(b,a)
    h = abs(h)
    return(w/max(w), h)
```

Make the impulse response function and use terminal formatted output (=doctest block.)

```
>>> n = 11.
>>> n
11.0
>>> b = repeat(1/n, n)
>>> b
array([ 0.09090909,  0.09090909,  0.09090909,  0.09090909,  0.09090909,
        0.09090909,  0.09090909,  0.09090909,  0.09090909,  0.09090909,
        0.09090909])
```

Calculate the frequency response and plot it:

```
w, h = mfreqz(b)
#Plot the function
plot(w,h,'k')
ylabel('Amplitude')
xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
show()
```

The first 10 values of the frequency response (w,h) as a table, notice that the code is hidden in the output document.

Amplitude	Frequency
1.0	0.0
1.0	0.0
1.0	0.0
1.0	0.01
1.0	0.01
1.0	0.01
0.99	0.01
0.99	0.01
0.99	0.02
0.98	0.02

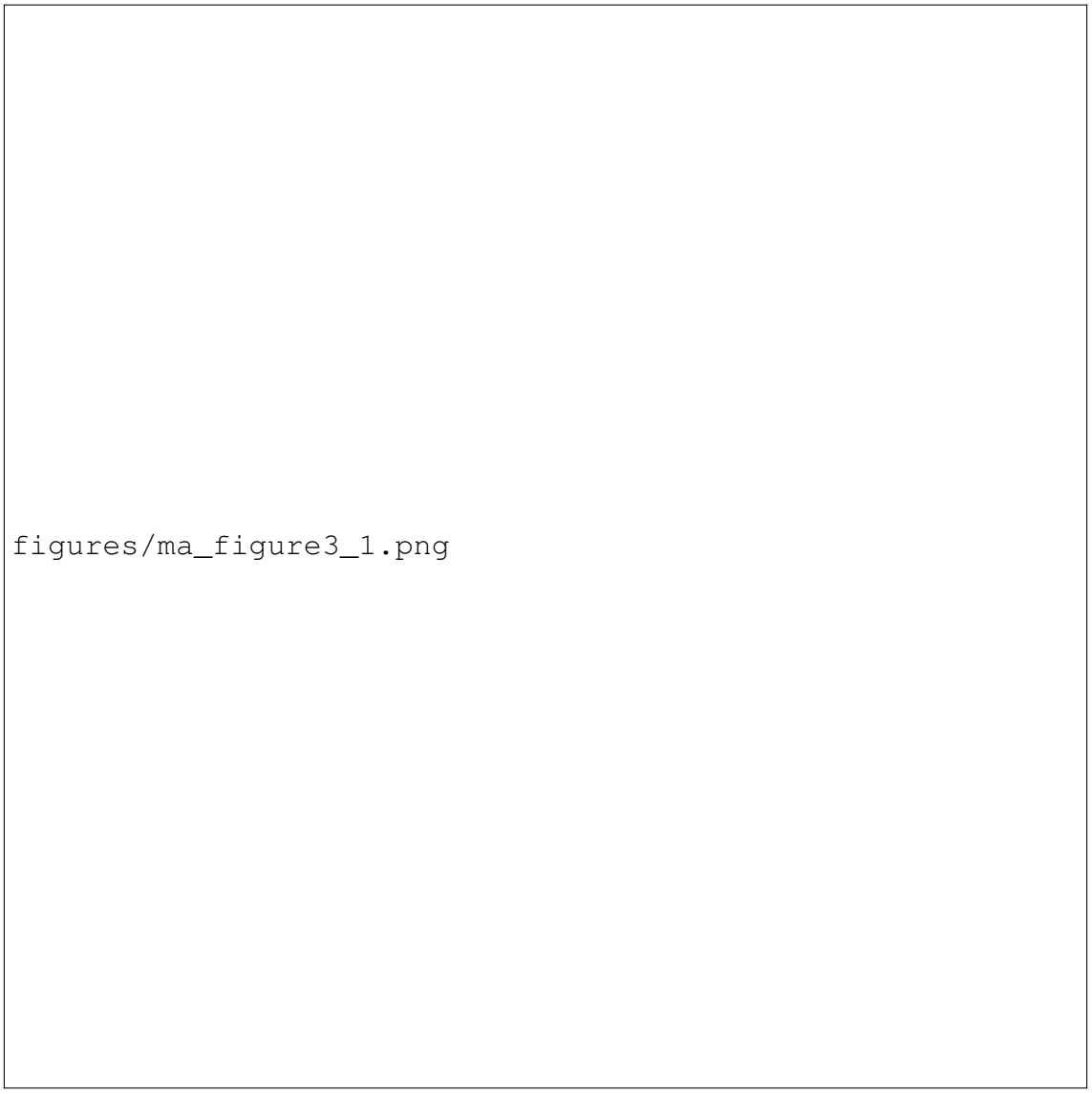

Figure 1: Frequency response of an 11 point moving average filter

7 LaTeX example

This is a simple example of a Pweave file (ma-tex.texw) that uses LaTeX as the documentation markup. The file demonstrates basic usage of Pweave and how it can easily be used to add dynamic figures and tables.

Processing the file with raw latex format:

```
Pweave -f tex ma-tex.texw
```

Processing the output .tex file with pdflatex produces this pdf

Processing the file with texminted format:

```
Pweave -f texminted ma-tex.texw
```

And the resulting pdf.

Note: You need to add `\usepackage{graphicx}` if you are including plots to document's preamble yourself and `\usepackage{minted}` if you are using minted.

7.1 Example Pweave document

And as result we get the LaTex document ma-tex.tex (shown below).

7.2 LaTeX output

8 Using Pweave with Emacs

For me the best option for editing Pweave files is Emacs using the noweb-mode. I use .Pnw for Pweave documents written with reST markup and .Plw for LaTeX markup. Here is what I have in my ~/.emacs.d/init.el to make Emacs recognize my Pweave documents correctly.

```
;Pnw-mode for Pweave reST documents
(defun Pnw-mode ()
       (require 'noweb-font-lock-mode)
       (noweb-mode)
       (setq noweb-default-code-mode 'python-mode)
       (setq noweb-doc-mode 'rst-mode))

(setq auto-mode-alist (append (list (cons "\\.Pnw$" 'Pnw-mode))
                    auto-mode-alist))

;Plw-mode for Pweave Latex documents
(defun Plw-mode ()
       (require 'noweb-font-lock-mode)
       (noweb-mode)
```

```
        (setq noweb-default-code-mode 'python-mode)
        (setq noweb-doc-mode 'latex-mode))

(setq auto-mode-alist (append (list (cons "\\.Plw$" 'Plw-mode))
                    auto-mode-alist))
```

The code simply sets the documenation mode (*noweb-doc-mode as rst-mode*) as reStructured-Text or LaTeX depending on the extension and the code mode as Python, so that the code chunks will be correctly formatted.

You can get the needed .el files and my full init.el from bitbucket: http://bitbucket.org/mpastell/emacs.d/src

9 Pweave links

Here are links to sites using Pweave. If you like to be included send me an e-mail.

- Nicky van Foreest: Python Code
- Python, Pweave and pandoc howto
- UCL GeogG122 Scientific Computing: Numerical and Scientific Python and Data Visualisation

10 Indices and tables

- *genindex*
- *modindex*
- *search*