Chapter 1

Heading 1 nb2pdf

Heading 1 is a LaTeX chapter.

Headings down to level 5 (LaTeX paragraph) are supported. Headings at level 6 are treated as normal body text. If you don't need a chapter (i.e., for an article), just don't use a heading at level 1.

When any level of section heading is formed, a label is automatically created from the text in the heading, removing all non-alpha characters. If the section heading is very long the the label will also be long, so keep it short. Also, don't use the same text twice as a heading.

1.1 Heading 2 Second-level heading

1.1.1 Heading 3

Heading 4

Heading 5 Heading 6

some more markdown.

1.2 The Header File

The `ipnb2tex.py` script encapsulates the IPython notebook code in some template LaTeX code. So the structure is as follows:

```
header
notebook contents
footer
```

The footer is the trivial `\end{document}` statement, the normal way to close a LaTeX file. The script adds `\end{document}` statement automatically, no action required on your part.

The default header is contained in the file `header.tex`, and write the LaTeX preample for an article document, with all the necessary package includes to support the notebook text. The header also adds the `\begin{document}` statement, and any 'standard template' entries after the `\begin{document}`. The first content added by the script is the first `\section{}` entry.

If all you need is a standard LaTeX report template document, then you can use the default header. If not, replace the default header file with your own.

An example of a special header file is given in this repo: `header-wp.tex`, which uses the `workpackage.cls` LaTeX doc style to format the latex with a project cover page. In the present version, you have to type the cover page variables into the header file, so it is not a generic header. Maybe later.....

1.3 The ./pic directory

The script saves the png files in the `./pic` directory, and the LaTeX code also expects to find the png files there. Hence, there must be an existing `./pic` directory in the directory where you run the script.

1.4 'Raw NBConvert' cells

The 'Raw NBConvert' cell type can be used to propagate text without any modification to the output stream. The IPython notebook will not touch this text; it is passed on as-is to the converter code. This this case we can add LaTeX code that will progress straight through to the LaTeX compiler.

Raw NBConvert cell. The could be raw LaTeX as in

$$f_{WN}(\theta;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} \exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{1.1}$$

The degree symbol in LaTeX should not be done with `\circ` as shown here: 45°. Rather load `\usepackage{gensymb}` package and use `45\textdegree{}` 45°, or `17~\celsius` 17 ℃.

1.5 Captions

There are three objects that can have captions: listings, figures and tables.

Figures and tables without captions are non-floating and appear in the text at the position it is found in the input file. Figures and tables with captions are floating.

Listings are always non-floating.

A single cell can contain any combination of captions, and these are indicated in the metadata field of the cell. Select the `Edit Metadata` drop-down option in the `Cell Toolbar` drop-down box (towards the right of the toolbar). This should expose metadata buttons on all the cells. Click on the button and type in the metadata.

The metadata has the following structure:

```
{
  "listingCaption": {
    "caption": "Comparison of effective drag parameters",
    "label": "lst:comparedrag",
    "outputCaption": ""
  },
  "figureCaption": {
    "caption": "Caption text for first figure",
    "label": "fig:lab1",
    "scale": "0.2",
    "width": "0.2"
  },
  "tableCaption": {
    "caption": "Caption text for first table",
    "label": "tab:lab1",
    "format": "{|p{10mm}|l|r|c|c|p{50mm}|p{20mm}|}",
    "fontsize": "normalsize"
  }
}
```

Note the placement of commas. No newlines are allowed in a single string in the metadata.

1.6 Python code

Python code blocks are printed using the LaTeX `listings` package. The format details can be defined in the header template file. These code listings can be supplied with a caption and a label(but these code blocks are not floating entities).

```
  {
  "listingCaption": {
    "caption": "Caption text for first listing",
    "label": "lst:listing1",
    "outputCaption": "Caption for first listing output"
  }
}
```

where the different fields have the following meanings:

1. `caption` specifies the text to be used in the listings caption. If this metadata field is not supplied, the listing will not be given a caption.
2. `label` specifies the label to be used for the listing, which can be used in LaTeX to reference the listing. The code block uses the label exactly as given in the metadata field. If the output listing is given an `outputCaption`, the output caption is labelled by the a label starting with the `label` string given, but appended with the string `-out` (i.e., `lst:listing1-out` in this case).
3. `outputCaption` specifies the label that is used for the output listing of the cell. If this metadata field is not supplied, the output listing will not be given a caption.

We can refer to Listings 1.1, 1.2, 1.3, and 1.4. These listings may not be defined in the IPython output but they are defined in the LaTeX output.

You can also refer to Figures 1.1 and 1.4.

Listing 1.1: Caption text for first listing

```
from IPython.display import display
from IPython.display import Image
from IPython.display import HTML

import os.path
import numpy as np
%matplotlib inline

import this
```

Listing 1.2: Caption for first listing output

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

The following code block has the metadata

```
{
  "listingCaption": {
    "caption": "Caption text for second listing",
    "label": "lst:listing2"
  }
}
```

which will give the code block a caption and a label, but the output is not captioned.

The code block also demonstrates how long lines are handled in this lstlistings configuration, with a linefeed symbol to mark the line.

Listing 1.3: Caption text for second listing

```
packCSArea = 2
packradius = np.sqrt(packCSArea / np.pi)
packDrag = 1.0
print(' ')
print('Pack radius = {}'.format(packradius))
print('Ppack frontal area = {} m2'.format(packCSArea))
print('Drag = {}'.format(packDrag))
print('Drag* area = {} m2'.format(packDrag * packCSArea))
print('This is a very long line of python code, that is supposed to flow over to the next line↵
    , in order to test the listing display')
```

```
Pack radius = 0.797884560803
Ppack frontal area = 2 m2
Drag = 1.0
Drag* area = 2.0 m2
This is a very long line of python code, that is supposed to flow over to the next line, in ↵
    order to test the listing display
```

Listing 1.4: Caption text for third listing

```
a = [0, 1]
for v in a:
    print(v)
```

```
0
1
```

The following code block has no metadata, and hence no captions on either the code or the output listings.

```
print('code with no listing caption')
```

```
code with no listing caption
```

Output emanating from different cells *may* result in different output listings in LaTeX even if they appear in a single output box in the notebook. The cells below attempt to recreate a similar scenario in a different application, but it does not seem to repeat the problem here.

```
def doprint(i):
    print('function output {}'.format(i))
    return int(i)
```

```
for i in [1,2,3]:
    si = str(i)
    v = doprint(si)

val = v * 3
print(val)
print(v)
```

```
function output 1
function output 2
function output 3
9
3
```

Figure 1.1: Caption text for first figure

1.7 Bitmaps

PNG images, read in from externally or created by Matplotlib, are imported into the LaTeX code with lines of the following form:

`\includegraphics[scale=0.2]{./pngs/test2LaTeX\_21\_0.png}`

where the scale determines the size of the image (obtained from metadata, see below), and the image file name is constructed from the notebook file name and cell information. The name is the concatenation of the notebook file name, the cell index (index number of the cell in the notebook file) and the sub-index image in the cell's output (there can be more than one image). All indices are zero based.

The scale or width is read from the metadata, which must be structured as follows:

```
{
  "figureCaption": {
    "caption": "Caption text for first figure",
    "label": "fig:lab1",
    "scale": "0.2",
    "width": "0.2"
  },
  "listingCaption": {
    "caption": "Listing caption in cell with a figure",
    "label": "lst:figurelisting",
    "outputCaption": "Output for a cell with a figure"
  }
}
```

Width is the fraction of LaTeX textwidth, i.e., if width=1, the graphic will be the full width of the text on the page. Width have precedance over scale and if width is given, scale is ignored.

The metadata also contains a caption string, and a label for the figure. The type must be `figure`.

If the `caption` field is given (or has non-zero length), the image will be written to a LaTeX float with a caption (and label). If the `caption` fields is not given, the image will not be encapsulated in the floating figure.

Listing 1.5: Listing caption in cell with a figure

```
#this figure is given meta data caption parameters, it will be floated in the latex export
display(Image(filename='images/keep-calm-and-code-python_BW.png', width=250, height=250))
print('also force output')
```

Listing 1.6: Output for a cell with a figure

```
also force output
```

The following figure is not floating, but is scaled down to be smaller.

```
{
  "figureCaption": {
    "scale": "0.3"
  }
}
```

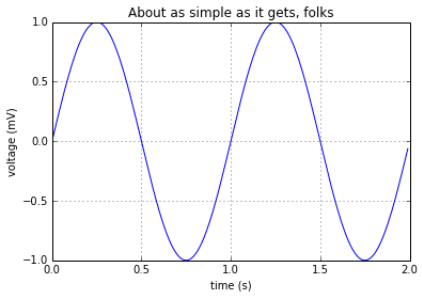
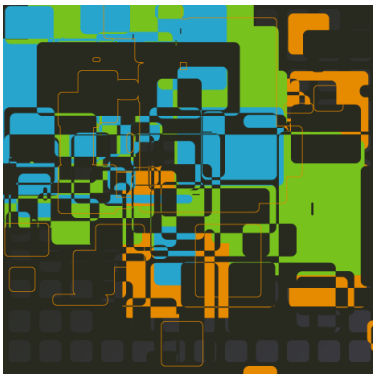


Figure 1.2: Caption text for third figure

```
#this figure is not given meta data caption parameters, it will be inlined in the latex export
display(Image(filename='images/random_squares_2.png', width=250, height=250))
```



The following Matplotlib plot is scaled and floating.

```
{
  "figureCaption": {
    "caption": "Caption text for third figure",
    "label": "fig:lab3",
    "scale": "0.4"
  }
}
```

```
%matplotlib inline

import pylab as pl
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)
pl.plot(t, s)
pl.xlabel('time (s)')
pl.ylabel('voltage (mV)')
pl.title('About as simple as it gets, folks')
pl.grid(True)
```

It is possible to have more than one plot in a single cell output. In this case each output can be made floating separately or be non-floated (you can even mix floating and non-floating). The captions are given in the metadata in the following format:

```
{
  "figureCaption": {
    "caption": "['Caption text for (first) fourth figure','','Caption text for (third) fourth figure']",
    "label": "fig:lab4",
    "scale": "[0.5, 0.5, 0.5]"
  }
}
```

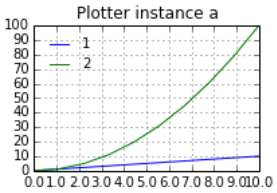


Figure 1.3: Caption text for (first) fourth figure

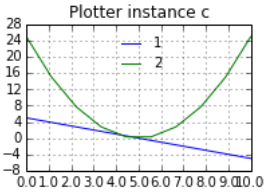
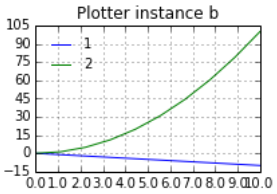


Figure 1.4: Caption text for (third) fourth figure

In this case the caption strings and scaling values are allocated to the sequence of images in the order given. If a caption string is zero length, that specific image will be non-floating. The float labels are determined from the root value given in the metadata, but with the output sub-index appended, as in `fig:lab4-0`, `fig:lab4-1`, `fig:lab4-2`.

```
import pyradi.ryplot as ryplot

x = np.linspace(0,10,10)
a = ryplot.Plotter(1,figsize=(3,2))
b = ryplot.Plotter(2,figsize=(3,2))
c = ryplot.Plotter(3,figsize=(3,2))
for i in [1,2]:
    a.plot(1,x,x ** i,'Plotter instance a',label=[str(i)])
    b.plot(1,x,(-x) ** i,'Plotter instance b',label=[str(i)])
    c.plot(1,x,(5-x) ** i,'Plotter instance c',label=[str(i)])
```



1.8 Cells with Python errors

```
type(eval("this is a test string")) is not list
```

```
  File "<string>", line 1
    this is a test string
                 ^
SyntaxError: invalid syntax
```

1.9 Embedded code (verbatim text)

Some firewalls are set up to grant `localhost` execution rights. In this case the server can be started with the command

```
ipython notebook --ip=localhost
```

Once started, the pages are served from

`http://localhost:8888/`

and not from `http://127.0.0.1:8888/`.

Embedded code meant for illustration instead of execution in Python:

```
def hello\_ipython():
    print "Hello IPython!"
```

1.10 Hyperlinks, references and citations

1.10.1 Embedded hyperlinks

The IPython website[1] is the central repository of all things IPython. There are some really nice videos[2] on YouTube.

For *URI references*, the `[IPython website](http://ipython.org/)` markup structure is read and the URI is used to create a citation label (`httpipythonorg`) and the URI (`http://ipython.org/`) is written to a new LaTeX bibtex file for the references in this notebook.

For *other types of references* a workaround is required. IPython is currently weak in the area of other types of references, because of limitations in the markup language. The `ipnb2tex.py` script makes provision for including bibtex entries, by embedding the complete bibtex entry in a metadata field.

Using the above approach creates a new bibtex file using the data in the notebook. Perhaps you might want to reference an existing bibtex file, mapping to the citation labels used in the notebook. The issue here is that the IPython notebook does not have access to your LaTeX bibtex file, so it does not know the citation references you want to use (the ones in your existing bibtex file).

The approach taken here is to provide a look-up translation table in this or any prior cell of the notebook, to translate citation references from the local name to your existing bibtex name. This is done in a cross reference dictionary that maps the names created internally to the names in your external bibtex file.

The metadata has two different fields, one to do the citation label mapping and the other to embed complete bibtex entries in the metadata. The metadata field must have the following format:

```
{
  "bibxref": {
    "httpipythonorg": "httpipythonorg",
    "httpswwwyoutubecomwatchvaIXED26Wppg": "httpswwwyoutubecomwatchvaIXED26Wppg",
    "httpsenwikipediaorgwikiWrappednormaldistribution":
            "httpsenwikipediaorgwikiWrappednormaldistribution"
  },
  "bibtexentry": {
    "wing2006computational": "@article{wing2006computational,  title={Computational thinking},
           author={Wing, Jeannette M},  journal={Communications of the ACM},  volume={49},
           number={3},  pages={33--35},  year={2006}}",
    "gracec": "@MISC{gracec,  author = {Grace Cathedral},  title = {The Cathedral Labyrinths},
           url = {http://www.gracecathedral.org/labyrinth/}}"
  }
}
```

where (1) the bibxref keys are the local names and the values are the names in your existing bibtex file and (2) the bibtexentry keys are the citation labels used in the notebook. In the above example the bibxref maps to the same names, because I am using the locally generated bibtex file. Normally you would use your bibtex database entries.

Note that the IPython notebook is somewhat finicky on the json format; each single entry in the above metadata must be on a single line (line-feeds inside the strings are not allowed — lines tend to be very long). Also, note the location of commas, there should be commas after all entries, except the last entry in a given scope.

The `ipnb2tex.py` script:

1. Loads/appends the bibxref translation table from any/all cells (if present).
2. Reads the `[]()` markup structure and then builds a citation reference from the URI (by removing some characters).
3. Create a bibtex entry for the reference (subsequently written to file).
4. Using bibxref, translates the local citation label to your existing citation label.

If you included bibtex items in the metadata, you can refer to them using the normal LaTeX notation. Test [3] and [4].

1.11 General markdown formatting

Markdown basics: lists, markup and code

- list item
- list item
- nested list item - not yet supported, neither are font attributes.
- *italics*
- **bold**
- `fixed font`

1. Enumerated list item 1.
2. Enumerated list item 2.

The markup language (or the converter) breaks if an itemized list and an enumerated list are immediately adjacent — we need to separate the lists by text with with a `<p/>`.

1.12 Tables

The table in this cell is rendered in LaTeX with the following metadata:

```
{
  "tableCaption": {
    "caption": "Caption text for first table",
    "label": "tab:lab1",
    "format": "{|p{10mm}|l|r|c|c|p{50mm}|p{20mm}|}",
    "fontsize": "normalsize"
  }
}
```

A complex HTML table with row spans and column spans:

Table 1.1: Caption text for first table

a	b				c	1
e	f	g		h	i	2
	j		k	l	m	3
n	o	p	q		r	4
s	t	u	v	w	x	5

The tables in this cell are rendered in LaTeX with the following metadata:

```
{
  "tableCaption": {
    "caption": "['Caption text for (first) second table','','Caption text for (third) second table']",
    "label": "tab:lab2",
    "format": "['{|p{20mm}|r|}','','{|c|l|}']",
    "fontsize": "['normalsize', 'tiny', 'Large']"
  }
}
```

Github flavoured markdown tables are supported in the IPython notebook (floating in LaTeX as Table 1.2):

Table 1.2: Caption text for (first) second table

This	is
a	table

second table (non-floating in LaTeX):

This	is
a	small table

PHP Markdown Extra is also supported (floating in LaTeX as Table 1.3):

Table 1.3: Caption text for (third) second table

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Both of these markup extensions require Python Markdown 2.4.1 to render HTML.

1.13 Math

Using math mode (anything between two dollar symbols) is interpreted as LaTeX math:

$$D_{KL}(P||Q) = \sum_i ln(\frac{P(i)}{Q(i)})P(i) \tag{1.2}$$

but the double dollar math environment does not always work.

The current version of the translator does work reliably with math mode between two dollar symbols. It may or may not work properly, depending on prior content in the cell. The reason for this is that the parsing of the math environment using two dollar symbols does not work correctly at the moment.

The probability density function for yaw angle is the wrapped normal distribution[5].

$$f\_WN(\theta;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \sum^{\infty} \_k = -\infty \exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right]$$

please use matching `begin{equation}` and `end{equation}`:

$$f_{WN}(\theta;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} \exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{1.3}$$

1.14 Known deficiencies

Many.

1. Mathematics enclosed in two dollar signs do not always render correctly, also affecting subsequent math rendering.
2. The template interface is very limited at moment; just a single file that is included as a single entity.
3. A config file is required to better define in one place the variables such as the path to the images, etc.
4. Multiple output listings may arise if the output is written by different cells (see the example above).

1.15 Python and module versions, and dates

From: Introduction to scientific computing with Python[6] we learn: "To encourage the practice of recording Python and module versions in notebooks, I've created a simple IPython extension that produces a table with versions numbers of selected software components. I believe that it is a good practice to include this kind of table in every notebook you create. To install this IPython extension, run:"

```
# you only need to do this once
\%install\_ext http://raw.github.com/jrjohansson/version\_information/master/version\_information.py
```

Now, to load the extension and produce the version table

```
%load_ext version_information
%version_information numpy, scipy, matplotlib
```

Software	Version
Python	2.7.7 (default, Jun 1 2014, 14:17:13) [MSC v.1500 32 bit (Intel)]
IPython	2.1.0
OS	nt [win32]
numpy	1.8.1
scipy	0.13.3
matplotlib	1.3.1
Sat Jul 19 12:10:21 2014 South Africa Standard Time	

Bibliography

[1] [Online]. Available: http://ipython.org/

[2] [Online]. Available: https://www.youtube.com/watch?v=aIXED26Wppg

[3] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[4] G. Cathedral, "The cathedral labyrinths." [Online]. Available: http://www.gracecathedral.org/labyrinth/

[5] [Online]. Available: https://en.wikipedia.org/wiki/Wrapped_normal_distribution

[6] [Online]. Available: http://nbviewer.ipython.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-0-Scientific-Computing-with-Python.ipynb