Notes template

with the `macros` package

Tadej Svetina*
MIT

June 2019

Contents

* `tadej@mit.edu`

1 Introduction

This is a sample document using the custom notes class (with the macros package). The primary aim of the notes class is to move all the boilerplate usually found in the preamble of the `main.tex` file to a reusable class file, making the actual document code shorter and easier to read.

Complementing that, the secondary aim of the notes class and the macros package is to create short commands for some frequently used things, cutting down on typing/tinkering time and improving readability.

Finally, as we have a reusable class and package, we might as well define some nice looking defaults. For this purpose, the class/package define

- a custom color palette, based on Paul Tol's vibrant scheme
- nice fancy theorem environments

In this document I showcase all of the above, as well as how to adjust fonts in a python graph to match those in the document. To use the class/package combo (with the usual defaults), just put both files in the root project repository, and add the following line to the top of the `main.tex` file

```
\documentclass{notes}
\usepackage[theorems]{macros}
```

Nothing else is needed!

2 The notes class

The class mainly takes care of adjusting the visual appearance of the document and importing some frequently used packages. It is based on scrartcl class from the KOMA-Script package. All options are passed down to scrartcl.

2.1 Fonts, title and headings

The fonts used in the document are Bitstream Charter for the serif font (using the mathdesign package), Fira Sans for sans serif (using the FiraSans package and scaled by a factor of 0.95 to match with Charter) and `Inconsolata` for monotype.

As you can see, title and headings have been modified slightly. The heading number has been put in the margin, and the title (and subtitle) are now set in normal font (as opposed to bold).

2.2 Layout

I don't like the default scrartcl layout, so I implemented my own. It uses the geometry package (not `\KOMAoptions`) to set the parameters, and **geometry should be used to implement any changes to the layout**. Here's how the layout is set by default:

`width` The current length of `\textwidth`, clipped to the length of between 2 and 3 alphabets. The `hmarginratio` is left at the default of 1:1

`height` The maximum of $\frac{8}{11.5}$ of `\paperwidth` and `\paperwidth` minus 2.5 inches. The `vmarginratio` is left at the default of 2:3

`marginparwidth` Minimum of 1 inch and

$$\tfrac{1}{2}(\texttt{\textbackslash paperwidth} - \texttt{\textbackslash textwidth}) - 2\texttt{\textbackslash marginparsep}.$$

It is computed after `width` is computed.

If you should do any changes to the layout (say, setting a different paper size), you should issue a `\recalcmargins` command to recalculate paper layout according to the formulas above.

Example 2.1: Changing paper size

Say you want to change the size of the paper to A4 (default is letter). To do this you put the following code in the preamble

```
\geometry{paper=a4paper}
\recalculatemargins
```

2.3 Code listings

To produce code listings, class uses the minted engine, which is wrapped inside a tcolorbox (to get rid of some spacing issues). The command for manual code input is `\tminted` and for code input from file `\tinputminted`; they both behave like their `t`-less counterparts.

Example 2.2: `tminted`

For `tminted` we can write

```
\begin{tminted}{python}
    print('1 + 2 = {}'.format(1+2))
\end{tminted}
```

which would output

```
print('1 + 2 = {}'.format(1+2))
```

To input code from a file – in this example code.py – we would write

```
\tinputminted[linenos]{python}{code.py}
```

which would produce

```
1  import numpy as np
2
3  def incmatrix(genl1,genl2):
4      m = len(genl1)
5      n = len(genl2)
6      M = None #to become the incidence matrix
7      VT = np.zeros((n*m,1), int)  #dummy variable
8
9      #compute the bitwise xor matrix
10     M1 = bitxormatrix(genl1)
11     M2 = np.triu(bitxormatrix(genl2),1)
12
13     for i in range(m-1):
14         for j in range(i+1, m):
15             [r,c] = np.where(M2 == M1[i,j])
16             for k in range(len(r)):
17                 VT[(i)*n + r[k]] = 1;
18                 VT[(i)*n + c[k]] = 1;
19                 VT[(j)*n + r[k]] = 1;
20                 VT[(j)*n + c[k]] = 1;
21
22                 if M is None:
23                     M = np.copy(VT)
24                 else:
25                     M = np.concatenate((M, VT), 1)
26
27                 VT = np.zeros((n*m,1), int)
28
29     return M
```

2.4 Loaded packages

The following packages are loaded by the class (apart from the font packages mentioned in 2.1):

- etoolbox
- calc
- inputenc, outputenc (with `utf8x` and `T1`)
- geometry

- enumitem
- footmisc
- amsmath, amsthm, IEEEtrantools
- graphicx
- booktabs
- tcolorbox (with `most` and `minted` options)
- printlen
- hyperref

3 The `macros` package

This package takes care of creating macros for commonly used things. Apart from this, it also defines a custom color palette, and an environment for (optionally) colorful theorems.

3.1 Package options

The macros package defines the following options:

`theorems` Creates the default theorem environments (see 3.5)

`nofancy` This removes the fancy formatting from the default theorem environments (if they are created), giving them the standard amsthm formatting

`numindep` This makes the default theorems be numbered independently (of sections), i.e., they each have their own counter.

3.2 Colors

The color palette defines the following colors (which are obtained by `m` + capitalized name of the color):

- **blue**, **cyan**, **teal**, **orange**, **red**, **magneta**, **grey**

There are two commands that use these colors: **`\alert`** and **`\define`**. Additionally, mainly for use with theorems, the following 4 light colors are defined (obtained by `l` + capitalized name of the color)

- **blue**, **cyan**, **mint**, **pear**

3.3 Math macros

First, some commonly used simple expressions are given short and easy to remember command names. They are put inside `\ensuremath`, so they can be safely used outside math mode, without being enclosed in `$$`. The commands are

```
\R, \E, \Q, \Z, \N, \Rp, \Rnn, \si,
```

and they produce

- $\mathbb{R}$, $\mathbb{E}$, $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{R}_{>0}$, $\mathbb{R}_{\geq 0}$, $\sum_{i=1}^{n}$

I also define the `eqns` and `eqns*` environments, which are just wrappers for the `IEEEeqnarray` environment with the `rCl` column layout.

Example 3.1: Multiline equations with eqns

A very simple multiline equation would be typset with

```
\begin{eqns*}
1 + 2 + \dots + n &=& \si i \\
                  &=& \frac{i(i+1)}{2}
\end{eqns*}
```

producing

$$\begin{aligned} 1+2+\cdots+n &= \sum_{i=1}^{n} i \\ &= \frac{i(i+1)}{2} \end{aligned}$$

3.4 Floats

There are two commands which simplify the usual inclusion of floats (tables and figures) — that is, they produced centered figures/tables. These two commands are:

```
\inclfig[pos]{filename}{caption}{label}
\incltab[pos]{filename}{caption}{label}
```

The `pos` argument is the usual float positioning argument, by default it is set to `htbp`. Mimicing the `float` package, you can use the `H` specifier, in the background `\captionof` is used (see Example 3.2), so that inline figures can be used inside the `tcolorbox` environments. If `label` is used, it will be prepended by `fig:` or `tab:`.

Example 3.2: An inline figure

Here's how we would include an inline figure:

```
\inclfig[H]{metro.jpg}{A metro train}{}
```

And here's how the result would look like:

Figure 1: A metro train

Example 3.3: Using booktabs

Here's how to make a simple table using the booktabs package:

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Table 1: Price list

The code used to produce this example is

```
\begin{tabular}{@{}llr@{}}
    \toprule
    \multicolumn{2}{c}{Item} \\
    \cmidrule(r){1-2}
    Animal    & Description & Price (\$) \\
    \midrule
    Gnat      & per gram    & 13.65      \\
              &    each     & 0.01       \\
    Gnu       & stuffed     & 92.50      \\
    Emu       & stuffed     & 33.33      \\
    Armadillo & frozen      & 8.99       \\
    \bottomrule
\end{tabular}
```

and it was included in the document using

```
\incltab[H]{table.tex}{Price list}{prices}
```

3.5 Theorems

The macros package provides support for creating colorful theorem (not limited to only theorems) environments, made with tcolorbox, to catch the reader's attention (nofancy is also an option). A few theorems can be defined by default (with the theorems option), and you have the option to create your own. Finally, the proof environment is also "packaged" in a colorful box.

3.5.1 Defining your own theorem environments

You can define your own theorem enviroment with

```
\newthem[fancy]{thmname}{title}{style}{numbering}
```

First, the thmname parameter sets the name of the new environment (as well as the \label prefix), while title sets the title of the environment (as will be displayed). The fancy argument can be set to nofancy — in this case the theorem is made with amsthm environments. The style argument controls the appearance — the color if the theorem is fancy, and the amsthm theorem style if it is not fancy. Finally, the numbering argument controls the numbering — if it is set to numindep the counter of the theorem will be independent of sections, otherwise it will be numbered withing sections.

Once we have defined a theorem, we can use it with

```
\begin{thmname}{title}{label}
Content
\end{thmname}
```

or

```
\begin{thmname*}{title}{label}
Content
\end{thmname*}
```

where title is an optional specific title, and label controls the label name (full reference will be thmname:label). The starred version, as usual, produces an unnumbered theorem.

Example 3.4: Fancy theorem

Let's create a fancy theorem called Alert, which should be red. We do this with

```
\newthem{alrt}{Alert}{mRed}{}
```

Then, let's create an alert with

```
\begin{alrt}{}{Danger}Watch out!\end{alrt}
```

which produces

Alert 3.1: Danger

Watch out!

Example 3.5: Boring theorem

Let's create a nofancy theorem called Proposition. We will style it with the `plain` amsthm style.

```
\newthem[nofancy]{prop}{Proposition}{definition}{numindep}
```

We can then create a proposition with

```
\begin{prop}{Boring}{}This is a proposition.\end{prop}
```

which produces

Proposition 1 (Boring)**.** This is a proposition.

3.5.2 Default theorems

The following theorems are defined if the `theorems` option is used (name of environment given in parentheses).

- Theorem (`thm`)
- Lemma (`lem`)
- Example (`exm`)
- Definition (`dfn`)

If the `nofancy` option is passed, these theorems will be formatted with the `nofancy` argument[1]. Their default (fancy) look of these environments is demonstrated bellow:

[1] Passing the `nofancy` option creates a `nofancy` boolean — you could use it in your own theorems, so that the look of all theorems will be defined with the `nofancy` option.

Theorem 3.1: Cauchy-Bunyakovsky-Schwarz inequality

For all vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in an inner product space it is true that

$$|\langle \boldsymbol{u}, \boldsymbol{v} \rangle|^2 \leq \langle \boldsymbol{u}, \boldsymbol{u} \rangle \cdot \langle \boldsymbol{v}, \boldsymbol{v} \rangle \tag{1}$$

Lemma 3.1: Titu's Lemma

For some $u_i, v_i \in \mathbb{R}_{\geq 0}$, it is true that

$$\frac{\left(\sum_{i=1}^{n} u_i\right)^2}{\sum_{i=1}^{n} v_i} \leq \sum_{i=1}^{n} \frac{u_i^2}{v_i} \tag{2}$$

Definition 3.1

We say that a set A is **closed**, if its complement A^C is open.

Example 3.6

In the indiscrete topology, the closed sets are X and $\emptyset$.

3.5.3 Proof

Without `nofancy` option, the default proof environment is put in a cyan box (while with the `nofancy` option, it is left as is).

Example 3.7: Proof

Let's write a simple proof:

```
\begin{proof}
    To prove Lemma \ref{lem:titu}, we just need to replace $u_i' =
    ↪ \frac{u_i}{\sqrt{v_i}}$ and $v_i' = \sqrt{v_i}$, and then
    ↪ apply Theorem \ref{thm:cs}.
\end{proof}
```

The output is:

Proof. To prove Lemma 3.1, we just need to replace $u_i' = \frac{u_i}{\sqrt{v_i}}$ and $v_i' = \sqrt{v_i}$, and then apply Theorem 3.1. □

4 Python graphs

When trying to match the style of python (matplotlib) graphs, two main adjustments have to be considered: fonts and colors.

4.1 Text width

But even before that, another "adjustment" should be made: the dimensions of the figure should be set to be smaller than those of the document. The relevant dimension here is width, or to be precise `\textwidth`. But the pure LaTeX ways of getting this give us the size in points (and we need it in inches), so to get it we use the `printlen` package:

```
\uselengthunit{in}\printlength{\textwidth}
```

As an example, the text width of this document is 5.56432 in.

4.2 Fonts

We need to make sure that the fonts in the document match the fonts in the graph. Matplotlib has a limited selection of fonts, so its safest to let LaTeX do all the font processing. We need to load the fonts and change the size to match that of our document. We would do this by putting this at the beginning of our code

```
from matplotlib import rc

rc('text.latex',preamble=r'\usepackage[charter, cal=cmcal]{mathdesign}')
rc('text',usetex=True)
rc('font', family='serif', size=11)
```

Note that this will slow down processing quite a bit, so **only change the fonts when the figures are done!**

4.3 Colors

We might also want to use our standard color palette (see section 3.2) – although this is more a matter of taste, there is nothing wrong with matplotlib color palettes. The way we would implement them is with

```
from matplotlib import rc
from cycler import cycler

colors = ['#0077BB', '#EE7733', '#009988', '#CC3311', '#33BBEE',
 ↪  '#EE3377']
```

```
rc('axes', prop_cycle = cycler(c=colors))
```

4.4 An example

Let's put the two things together, to create a simple example. Here is the code:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
from cycler import cycler

# Set up fonts
rc('text.latex',preamble=r'\usepackage[charter, cal=cmcal]{mathdesign}')
rc('text',usetex=True)
rc('font', family='serif', size=11)

# Set up colors
colors = ['#0077BB', '#EE7733', '#009988', '#CC3311', '#33BBEE',
 ↪  '#EE3377']
rc('axes', prop_cycle = cycler(c=colors))

# Plot
x = np.linspace(-np.pi, np.pi, int(1e3))
fig, ax = plt.subplots(figsize=(5,4))

for i in range(6):
    ax.plot(x, np.sin(x + i*np.pi/6))

# Label graph
ax.set_title(r'$f(x) = \sin\left(x + \frac{k}{6}\pi\right)$'+\
             r' where $k \in \{0,1,2,3,4,5\}$', fontsize=11)
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$f(x)$')

# Tweak layout and save figure
plt.tight_layout()
fig.savefig('../../Dropbox
 ↪  (MIT)/Apps/Overleaf/Templates/Notes/trig.pdf',
            pad_inches=0.1, bbox_inches='tight')
```

The result is shown in Figure 2. Note that we set the width of the figure to 5 inches – that's ok, as we have seen that the width of our document is 5.56432 in.

There are also a few extra useful tweaks in the code:

- The title size is set, in line 25, to the document size (11) – this is because by default, the title size is a bit bigger (other graph elements do not have this problem). But this is most likely not relevant anyways, as you would not need a title on a figure that will already have a caption.
- The path in `savefig` (line 31) is set to the Overleaf Dropbox repository. This is a great way to get continuous updating with Overleaf every time you change the figure in the code, without having to manually copy/paste or git commit/push.

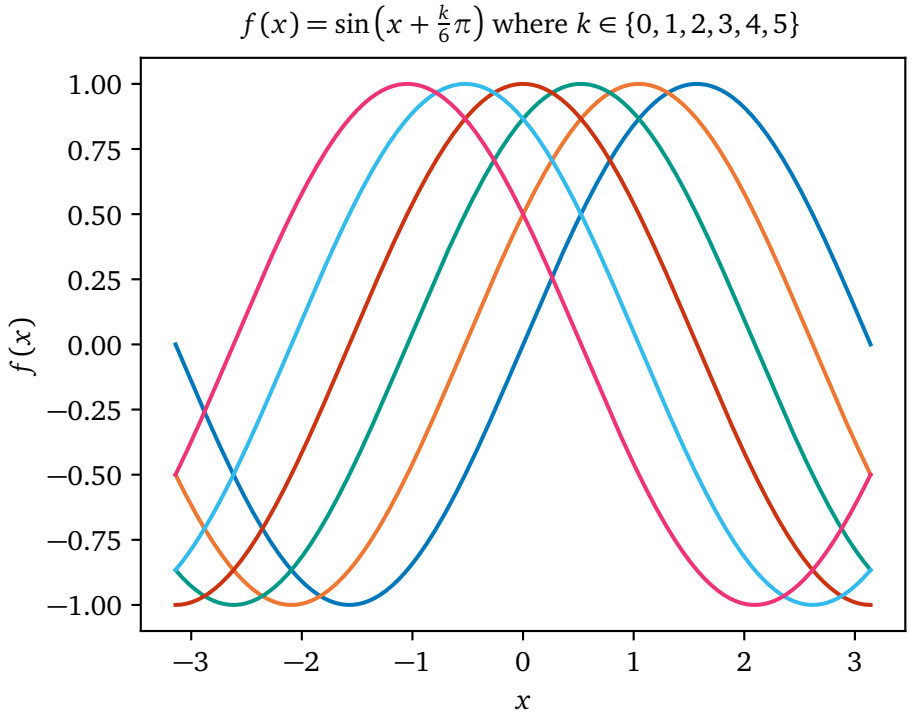

Figure 2: Some trigonometric functions

5 Todo

There are a couple of things that I would like to add to this class (package), including

- Macros for some commonly used list variations from the enumitem package