MLR How Tos

Marc Toussaint

September 7, 2011

Contents

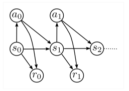
1 LaTeX Conventions

1.1 References

```
\usepackage[round]{natbib}
\bibliographystyle{abbrvnat}
```

1.2 Generating figures / graphical models

(i) Use Inkscape. See the example mdp.svg in the pics path (note the `\graphicspath{{pics/}}` command):



Use the 'Tex Text' plugin. Use layers to generate animations. Use 'save a copy' to export as pdf (potentially with some layers switched off)

(ii) Use xfig with 'special tag' text. Use the following fig2pdf script to convert:

```
rm -f $1.pdf
rm -f $1.tex
fig2dev -Lpdftex -p $* $1.fig $1.pdf
fig2dev -Lpstex_t -p $* $1.fig $1.tex

cp $HOME/write/tex/figs/header.tex z.tex
printf "%s\n" "\\input{$1.tex}" >> z.tex
printf "%s\n" "\\end{document}" >> z.tex
more z.tex
pdflatex z.tex
pdfcrop z.pdf $1.pdf
```

1.3 Space tricks

Be creating in using any of the following

```
  \renewcommand{\baselinestretch}{.98}
  \renewcommand{\arraystretch}{1.2}
  \renewcommand{\textfloatsep}{3ex}
  \renewcommand{\floatpagefraction}{.6}
  \renewcommand{\dblfloatpagefraction}{.6}
  \setlength{\mathindent}{2.5em}
  \setlength{\jot}{0pt} %zwischen den math zeilen
  \setlength{\abovedisplayskip}{-10pt}
  \setlength{\belowdisplayskip}{-10pt}
  \setlength{\mathsurround}{-10pt}
  \renewcommand{\floatsep}{-1ex}
  \renewcommand{\topfraction}{1}
  \renewcommand{\bottomfraction}{1}
  \renewcommand{\textfraction}{0}
  \columnsep 5ex
  \parindent 3ex
  \parskip 1ex
```

Or more compact lists:

```
\begin{list}{--}{\leftmargin4ex \rightmargin0ex \labelsep1ex
  \labelwidth2ex \topsep-\parskip \parsep.5ex \itemsep0pt}
\item ...
\item ...
\end{list}
```

The `list` parameter documentation:

```
   * \topsep amount of extra vertical space at top of list
   * \partopsep extra length at top if environment is prececed by a blank line (it should be a rubber l
   * \itemsep amount of extra vertical space between items
   * \parsep amount of vertical space between paragraphs within an item
   * \leftmargin horizontal distance between the left margins of the environment and the list; must be
   * \rightmargin horizontal distance betwen the right margins of the enviroment and the list; must be
   * \listparindent amount of extra space for paragraph indent after the first in an item; can be negat
   * \itemindent indentation of first line of an item; can be negative
   * \labelsep separation between end of the box containing the label and the text of the first line of
   * \labelwidth normal width of the box containing the label; if the actual label is bigger, the natur
   * \makelabel{label} generates the label printed by the \item command
```

1.4 Generating pseudo code

See Algorithm 1

Algorithm 1 Gauss-Newton with adaptive Levenberg Marquardt parameter

```
Input:  start point $x$, tolerance $\delta$, routines for $x \mapsto (\phi(x), J(x))$
Output: converged point $x$
 1: initialize $\lambda = 1$
 2: compute $(\phi, J)$ at $x$ and $l = \phi^\top \phi$
 3: repeat
 4:   compute $\Delta$ to solve $(J^\top J + \lambda \mathbf{I})\,\Delta = -\sum J^\top \phi$
 5:   $x' \leftarrow x + \Delta$
 6:   compute $(\phi, J)$ at $x'$ and $l' = \phi^\top \phi$
 7:   if $l' > l$ then
 8:     $\lambda \leftarrow 2\lambda$
 9:   else
10:     $\lambda \leftarrow 0.8\lambda$
11:     $x \leftarrow x'$
12:   end if
13: until $|\Delta| < \delta$
```

2 Coding Conventions

(i) Take the Google C++ style as default: `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`. The rules below overwrite Google conventions.

(ii) Output parameters before input parameters in function declarations.

(iii) Naming convention `name_of_a_routine(ouput,input);` is perhaps better than my old `nameOfARoutine(..);` style.

(iv) Make headers as clean and short and concise as possible!

(v) Try to avoid `#includes` in header files as much as possible!! In particular, when a class's methods require an external library; this external library should be included only in the cpp-file.

$\rightarrow$ In particular, try to avoid `#includes <external_lib>` in headers!! (I tried to get rid of them as much as possible in all my *.h)

(vi) Try to avoid `#includes` in header files as much as possible!! If you think the class needs to contain members/data structures defined in an external library and therefore you need to include its header in your header — that's often not true. Instead, hide all members in a *"hidden self"*:

Bad example:

```
//h-file:
#include <OpenCV> //BAD

struct MyClass{
  OpenCV_DataStructure data;
};
```

Good example:

```
//h-file:

struct sMyClass; //forward declaration of a 'hidden self' that will
contain all members hidden from the header

struct MyClass{
  sMyClass *s; //maybe call it 'self' instead
};

//cpp-file:
#include <OpenCV> //GOOD

struct sMyClass{
  OpenCV_DataStructure data;
};

MyClass::MyClass(){
  s = new sMyClass;
}
```

(vii) Move documentation to cpp files. Advantages: You can write as long and lengthy documentation as you want without destroying the beauty of the header. Doxygen will compile this without problem to provide a nice documentation. If people want to read the documentation from source directly—it's not much of a hassle to find the definition in the cpp file.

(viii) Never ever use `#ifdef` directives in a header file if this influences definition of classes, especially which members (and 'size') a class has!

The following debugging horror may happen: You define a class to have different members depending on a compiler flag. You compile your library with some compiler flags. The user includes your header with other compiler flags. Everything seems to compile and link fine. But when the user accesses members of the class, he actually refers to different memory addesses as your routines in your library.

Therefore try to avoid `#ifdefs` in headers as much as possible! Move them to the cpp file!

(ix) If you don't have a preferred IDE, use kdevelop.

K&R formatting conventions!

Editing: use spaces instead of tabs. 2 characters. Identation with 2 characters.

Keys: F10,11,12: step, step in, step out, F9: toggle break
F8 clean, F7 build, F5 debug run, CtrlF5 run, F6 continue

(x) fltk lists their conventions – I like them, also their style of formatting and their makefile conventions