LaTeX Reference

Caleb [*]

April 4, 2021

Abstract

We include brief descriptions of commonly used features encountered when writing documents and papers in LaTeX.

Contents

1 Ground Zero (AA)

I've included an empty LaTeX file and Makefile. In my Makefile, I do a couple unconventional things. First, I make the input and output name differently. I like to call my main file "main.tex" but rename output since Adobe has a bug they seem to cache similarly named files. I also break large TeX files into chapters by saving them as separate files and saving them in "chpts".

[*]Email: calebju@gmail.com

The choice of font is a metter of personal taste. I have enjoyed this forum on comparing fonts [link]. I do like the standard Modern font (`lmodtern`) or Utopia, the latter which is added by `usepackage[utopia]{mathdesign}`.

Now onto specifics for the LaTex file. Before we can use theorems, we have to explicitly define them.

Defining Theorem Macros

```
\theoremstyle{plain}
\newtheorem{theorem}{Theorem}[section]
\newtheorem{corollary}{Corollary}[theorem]
\newtheorem{lemma}[theorem]{Lemma}
\theoremstyle{definition}
\newtheorem{definition}{Definition}[section]
\theoremstyle{remark}
\newtheorem*{remark}{Remark}
```

Let's see how the *theorem* macro is define. The first parameter is the LaTeX keyword. The second is the text that appears in the PDF. The third is the counter. Notice we can *theorem*'s keyword as a counter. The location of where we include the counter impacts the numbering; if it included in the middle, the macro (e.g. *lemma*) uses the same numbering. If the counter is at the end (e.g. *corollary*), it is treated as a subheading to theorem and thus will reset with *theorem* (I don't know why I would ever do this).

Also, we can define macros with different styles, with the three default ones used above (see here for more information). In particular, theorem (denoted as "plain") includes both bolded and italicized text. Definition includes just bold. Remark is just bold. I find these to be sufficient, but the previous link shows how to define new ones. I often use plain for imporatnt results, definition for also examples and exercises, and remark for notes.

Theorem 1.1 (Fermat's Last Theorem)**.** *Given three positive integers* a, b, *and* c,

$$a^n + b^n = c^n.$$

Definition 1.1. This is a definition.

Remark. This is a remark.

The proof package is automatically included

Corollary 1.1.1. *If* $a = b = c = 0$, *then the theorem above holds for any* $n \in \mathbb{N}$.

Proof. Proof is easy. Observe $0^n + 0^n = 0^n$. $\square$

Notice that QED symbol is an empty box. We can fill it in. We can also bold "Proof" rather than italicize.

Modifying QED

```
\renewcommand\qedsymbol{$\blacksquare$}
\renewenvironment{proof}{\noindent {\bfseries Proof}}{\qed}
```

Lemma 1.2. *When* a, b, *and* c *are the side lengths of a right triangle, then*

$$a^2 + b^2 = c^2$$

Proof. Draw a triangle. You shall see it. ∎

Something very useful, especially during the editting phase, is to see the labels of specific equations and images. To that end, we can visualized these in the PDF itself with the useful packcage, which is included in the `amsmath` package.

Imports

```
\usepackage{showkeys}
```

1.1 Commands and Environments (AA1)

Imports

```
\usepackage{comment}
```

Let's now briefly overview commands and environments, which we have already used earlier. Examples of commands include `textbackslash {input}`. There are three ways to define commands.

New commands

```
\newcommand{\B}[1]{\mathbb{#1}}
\renewcommand*{\labelitemi}{\dag}
\providecommand{\caleb}[1]{\textcolor{blue}{Caleb:
 ↪  \textcolor{blue}{#1}}
```

The new command defines a new command and throws an error if it is already defined. Renew overwrites an existing command or throws an error if it is not defined. Finally, the provide command defines a new command if it doesn't exist. Notice in each command, we can include parameters and such. I'll include this link for more information, but the above is usually sufficient.

While commands are suitable for smaller portions of text, environments are are designed to handle blocks of LaTex code. You may already be familiar with some environments, such as `\begin{equation}`. Let's see how to create a simple environment with a numbering system

New commands

```
\newcounter{example}[section]
\newenvironment{example}[1][]{\refstepcounter{example}\par\medskip
   \noindent \textbf{Example~\thesection.\theexample. #1}
     ↪  \rmfamily}{\medskip}
```

Example 1.1. This is an example of an example.

Note that we can also create environments that can hide. This is useful if you want to create solutions. For theis, we will make use of the comment package, as shown below.

New commands

```
\newtheorem*{sol}{Solution}
\newif\ifshow
% \showfalse
\showtrue % swap with above to hide
\ifshow
    \newenvironment{solution}
    {\begin{sol} \color{gray}}
    {\end{sol}}
\else
    \excludecomment{solution}
\fi
```

Solution. This is the solution to blah blah

There is apparently a way to set show to true or false during make. I've tried this solution but it does not work with the prescribed solution. If interested, try to dig more into this.

2 Links and Formatting (BB)

Imports

```
\usepackage{hyperref} % for linking
\usepackage{bm}       % for bolding math
\usepackage{amsfonts} % for special characters
\usepackage{bbold}    % for calligrahic digits, e.g. \mathbb{1}
```

Let's say we have the following equation,

$$\begin{aligned} \min \quad & \|\beta\| \\ \text{s.t.} \quad & y_i(\beta^T x_i + \beta_0) \geqslant 1 - \zeta_i, \ i = 1, \ldots, N \\ & \zeta_i \geqslant 0, \ \sum_{i=1}^{N} \zeta_i \leqslant Z. \end{aligned} \tag{1}$$

eq:example1

We can reference the above equation using the LaTeXcommand `ref`, but by itself it will not cross-reference to the equation above (try removing the package). Instead, we must use the package `hyperref` and we can now link equation (1). eq:example1

Individual Label

```
\begin{align}
  \min & \hspace{10pt} x_1 + x_2 + \ldots + x_n \nonumber \\
  \text{s.t.} & \hspace{10pt} a_{i,1}x_1 + a_{i,2}x_2 + \ldots
      + a_{i,n}x_n \stackrel{(1)}{\leq} 1, \ i = 1,\ldots,m
   ↪  \label{eq:packing_constraints} \\
              & \hspace{10pt} x_i \geq 0, \ i = 1,\ldots,n
               ↪  \label{eq:nonnegative_constraints}.
\end{align}
```

We can also reference individual equations as well,

$$\begin{aligned} \min \quad & x_1 + x_2 + \ldots + x_n \\ \text{s.t.} \quad & a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n \overset{(1)}{\leqslant} 1,\ i = 1, \ldots, m \qquad (2) \\ & x_i \geqslant 0,\ i = 1, \ldots, n. \qquad (3) \end{aligned}$$

eq:packing_co

eq:nonnegativ

Equation (2) eq:packing_constraints are the packing constraints and equation (3) eq:nonnegative_constraints are the non-negativity constraints.

We can write this in matrix form. To get the bold font, we need the package `bm` and for blackboard fonts (special symbol for reals) we utilize packages `amsfonts` and `bbold` for digits. Note that we can also use the package `bbm`, however this only contains formatting for 1 and 2. We apply it below,

$$\begin{aligned} \min \quad & \mathbb{1}^T x \\ \text{s.t.} \quad & \mathbf{A}x \leqslant \mathbb{1} \\ & x \geqslant . \end{aligned}$$

3 Images (CC)

Imports

```
\usepackage{graphicx}   % input images
\usepackage{wrapfig}    % enable small figures
\usepackage{float}      % enable "H" placement
\usepackage{subcaption} % enable subimages/subcaptions
```

We start with a very basic image, which requires the `graphicx` package. The image is summoned with the code `\includegraphics[...]`. We specify the size of the image in the brackets, which can be absolute sizes (e.g. "3cm") or relative sizes (e.g., "0.5 \textwidth", see here for more references). Additionally, we can the image using via `\begin{figure}[X]`, where the variable X can be "h", "t", "b", "p", or "H". These, respectively, stand for placing the image *approximately* at the same point in the source code, at the top of the page, at the top of the page, at the bottom of the page, put on a special page for image floats (likely won't be used except appendix of images), or exactly where the source code. We can add an "!" (e.g., "h!") to tell LaTeXto override the internal parameters for "good" positions. Often, `h!` is equivalent to `H`, so use the former if the compiler complains (this is fixed by adding `\usepackage{float}`). Second, the image is not centered. To center it, we add the parameter `\centering` with in the figure blocks. Third, we can add captions by include `\captions`. The ordering of these commands is important. If we place the caption above the image, the caption will be above the image, and equivalently it will be below. Altogether, we get the following image.

Input an Image

```
\begin{figure}[h!]
\includegraphics[width=6cm]{figs/Saddle_point}
\centering
\caption{Saddle point problem, $f(x,y) = x^2 + y^2$}
\end{figure}
```

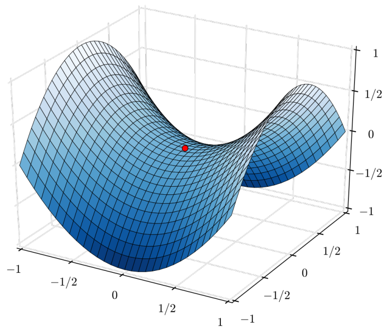


Figure 1: Saddle point problem, $f(x,y) = x^2 + y^2$

Note that it is possible to place the caption to the left or right using additional packages. See this link. Like equations and theorems, image can be referenced via `\label{fig:ref_name}`. Finally, here is a brief note on formatting the size and positioning of the figure.

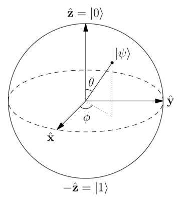
When the image is small, we may want to wrap it alongside text. To do so, we utilize the command `subfigure` instead of `figure` and configure its position and size and we must include `\subcaption`. While using `wrapfigure`, be wary of page breaks. One might have issues where the image is *overlapped* with the text, see here. Personally, I've found that if the `wrapfigure` must be between two texts of line. Without the text "Random" below, the image would be incorrectly placed.

Input an Image

```
\begin{wrapfigure}{r}{0.15\textwidth}
    \centering
    \includegraphics[width=0.15\textwidth]{figs/unit}
\end{wrapfigure}
```

Random.

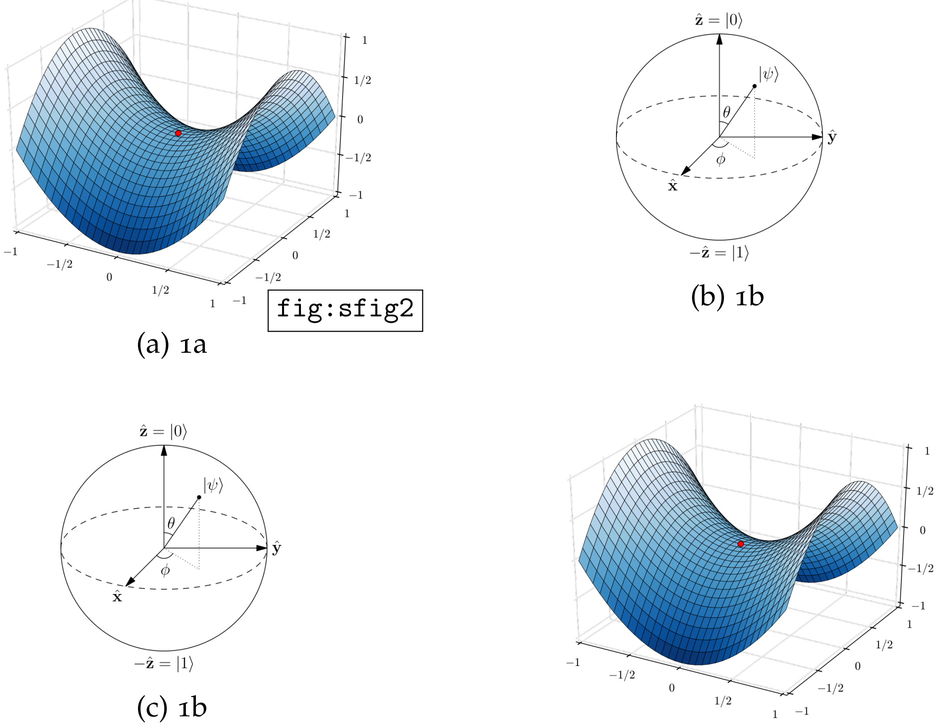
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



We can also include multiple figures aligned. We utilize the `figure` and within each figure will use `\subfigure`. This is a mechanical and so we just write the code below; copy when it is needed.

Input an Image

```
\begin{figure}[H]
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=.8\linewidth]{figs/Saddle_point}
  \caption{1a}
  \label{fig:sfig1}
\end{subfigure}%
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=.40\linewidth]{figs/unit}
  \caption{1b}
  \label{fig:sfig2}
\end{subfigure}
\\[\smallskipamount]
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=.40\linewidth]{figs/unit}
  \caption{1b}
  \label{fig:sfig2}
\end{subfigure}
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=.8\linewidth]{figs/Saddle_point}
  \caption{1a}
  \label{fig:sfig1}
\end{subfigure}%
\caption{Four figures in one}
\label{fig:fig}
\end{figure}
```



(a) 1a fig:sfig1 (b) 1b fig:sfig2 (c) 1b fig:sfig2 (d) 1a fig:sfig1

fig:fig

Figure 2: Four figures in one

4 Tables (DD)

Imports

```
\usepackage{booktabs}
```

There is a bit of art to making tables. This dichotomy is displayed in the first couple of slides in this presentation. In general,

1. Avoid vertical lines (lines may also be referred to as “rules”)
2. Avoid double lines. Instead, opt for thickness.
3. Put units in the headers (not in the body of the table)
4. Always preceed a decimal point by a digit; e.g., 0.1 instead of .1
5. Avoid “ditto” signs or any other such convention to repeat a previous value. In many circumstances a blank will serve just as well. If it won’t, then repeat the value.

With these guidelines in mind, we now introduce the two main components to creating tables: `tabular` and `booktabs`. The first package creates the table and the second package simple draws good (horizontal) lines to help organize the data. We start by briefly explaining `tabular`.

A table is invoked by the command \{ `tabular` }`[pos]{cols}`. The parameter `pos` is similar to "h" and "H" in figure. The second parameter `cols` explains how many columns we want, what the alignment of each column is, and also enables additional spacing for consecutive tiles. The code below shows an example, but a more detailed explanation can be found here.

The second relatively simple package, `booktabs`, is about nice looking horizontal lines. The three main commands one will utilize are `\toprule`, `\midrule`, and `\bottomrule`. These will usually be placed beginning with the toprule, then separating headings and data with the thinner midrule, and closing the table with bottomrule. These are placed after the \\of the last corresponding row. We can also subdivide columns in categories by separating the two rows using `\cmidrule`. A short pdf details more information to these commands. See the code below for a concrete example.

Create Table

```
\begin{table}[H]
\centering
% \begin{tabular}{@{\extracolsep{1cm}}l@{}lr@{}} \toprule
\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 \\
\addlinespace[5pt]
Total & & 150.10 \\ \bottomrule
\end{tabular}
\caption{This is a caption for the table}
\end{table}
```

Item		
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99
Total		150.10

Table 1: This is a caption for the table

Alternatively, here is a list of example tables.

5 Cool Styling (EE)

Imports

```
\usepackage{xcolor}
\usepackage{soul}
\usepackage{tcolorbox}
\tcbuselibrary{skins, breakable}
\usepackage{enumitem}
```

5.1 Text customization (EE1)

Like in word processing files, we would like to customize texts beyond the simple *italization* and **bolding**: things like coloring, underlining, strikethrough, hights, and more. We start with coloring. Start by importing `\usepackage{xcolor}`. From here, colors can be invoked by either `\textcolor{colorname}{Text}` or `{\color{colorname}Text}`. By default, the colors include yellow "base colors" such as blue and orange. These colors can be mixed by taking percentages of a color, suchas `green!55!blue`, a color that arises by taking 55% green and what remains as blue (45%). More colors can be obtained by including xcolor's different package options via `\usepackage[optionnames]{xcolor}`. Option names include `dvipsnames`, `svgnames`, and `x11names`. Their specific colors can be found in the documentation.

Next, we explore the world of highlighting by invoking the macro `\hl{}` after importing, `\usepackage{soul}`. We can choose colors other than yellow by running the command `\sethlcolor{colorname}`. Hightlighting and text-coloring can be combined by nesting the `hl` command within a `textcolor` or `color` command.

We now move onto styling with lines, such as strikethroughs and underlines. The package `soul` already includes strikethroughs and underlines via `\st{}` and `\ul{}`, respectively. Other customizations, such as changing the the color of the lines, applying all caps, and spacings can be found in the package's documentation.

5.2 tcolorbox (EE2)

The package `tcolorbox` is an *extensive* package that allows beautiful displays of boxes to showcase important results/text/ideas. There are too many features to consider learning at once (or demo-ing for that matter – the documentation is 530 pages long), so we only include the critical detail. The base package is included via `\usepackage {tcolorbox}`. By itself, tcolorbox is boring. We can extend it by including additional programs via `\tcbuselibrary{⟨keylist⟩}`. I typically include only `skins`, `theorems`, and `breakable`, but a fully laundry list can be found here.

We can start with a barebones box.

Input an Image

```
\begin{tcolorbox}
  This is a \textbf{tcolorbox}.
  \tcblower
  Here, you see the lower part of the box.
\end{tcolorbox}
```

This is a **tcolorbox**.

Here, you see the lower part of the box.

The options (the stuff inside the brackets) enables additional features, such as the color and fill of the box, the title, shadows, and many other features. I like the following setup the most so I'll only include this one.

Input an Image

```
\begin{tcolorbox}[colback=blue!5,colframe=blue!40!black,
                  title=Example Title, drop shadow,
                  skin=enhanced, fonttitle=\bfseries, breakable]
  This is an example box.
\end{tcolorbox}
```

Example Title

This is an example box.

Notice in the above box we have included the `breakable` option. For larger boxes, this allows the box to *cross pages*. An earlier version was used for the subfigure code.

Now, we can set the default baseline tcolorbox-es using `tcbset`. The application will be applied to all subsequent tcolorbox-es.

Input an Image

```
\tcbset{ drop shadow, skin=enhanced, fonttitle=\bfseries, }
\begin{tcolorbox}
  This is again a \textbf{tcolorbox}.
  \tcblower
  This was set using \texttt{tcbset}.
\end{tcolorbox}
```

This is again a **tcolorbox**.

This was set using `tcbset`.

I consider these to be the essential setup for `tcolorbox`. More features, such as fitting the box to the exact size of the LaTeX object, defining new boxes, followed by a full list of options, can others can be found starting here here.

To end the quick tutorial on tcolorbox, here is a quick STOP sign that can be drawn with it.

One more cool styling, which is useful in practice. We can try a nice gray box around an important result to highlight it.

Input an Image

```
\newcommand{\greybox}[1]{ \begin{tcolorbox}[sharp
    corners,colback=black!10!white,colframe=black!10!white]#1\end{tcolorbox}
 ↪  }
```

Lemma 5.1 (Three point lemma). *Let* $\{x_t\}$ *be generated by* (**??**) *with the Euclidian distance replaced by a Bregman divergence* D_ω*. Then for any* $\mathbf{x} \in \Omega$

$$\gamma_t \langle g(\mathbf{x}_t), \mathbf{x}_{t+1} - \mathbf{x} \rangle + D(\mathbf{x}_{t+1}, \mathbf{x}_t) \leqslant D(\mathbf{x}, \mathbf{x}_t) - D(\mathbf{x}, \mathbf{x}_{t+1}).$$

5.3 Lists (EE3)

The popular commands, `enumerate` and `itemize`, can be further customized if we include the package via `\usepackage{enumitem}` (with corresponding documentation found here). First, enumerate, we can customize which symbols are used to list each item via the `label` option (see the code below). We can set this to either `\alpha`, `\Alpha`, `\arabic`, `\roman`, or `\Roman`. Make sure to include an asterik at the end (I don't know what it does, all I know is that it doesn't compile otherwise).

Input an Image

```
\begin{enumerate}[label=\textit{\arabic*})]
  \item blah
  \item blahh
  \item bblahh
\end{enumerate}
```

1) blah

2) blahh

3) bblahh

We can do other customizations. To get a "tight" enumerate (i.e., less spacing), we can add additional options (I will not write it, instead see the code). Second, we can redefine the nested itemizations to have unique symbols (typical of what you would encounter in a word document). To do so, one needs to redefine the macros \labelitemi to \labelitemiv (see the code below). And as always, there is more code here. We display the end result below.

Input an Image

```
\renewcommand{\labelitemi}{$\bullet$}
\renewcommand{\labelitemii}{$\cdot$}
\renewcommand{\labelitemiii}{$\diamond$}
\renewcommand{\labelitemiv}{$\ast$}
%
%
\begin{enumerate}[label=\roman*), topsep=0pt, itemsep=-1ex,
 ↪  partopsep=1ex, parsep=1ex]
  \item {\texttt{\textbackslash topsep}}: space between first
    ↪  item and preceding paragraph.
  \item {\texttt{\textbackslash partopsep}}: extra space added
      to {\texttt{\textbackslash topsep}} when environment
    ↪  starts a new paragraph
        \begin{enumerate}[nosep]
          \item Is
          \begin{enumerate}[nosep]
                  \item this
                  \begin{enumerate}[nosep]
                          \item Inception?
                        \end{enumerate}
                 \end{enumerate}
        \end{enumerate}
  \item {\texttt{\textbackslash itemsep}}: space between
    ↪  successive items.
\end{enumerate}
```

i) \topsep: space between first item and preceding paragraph.
ii) \partopsep: extra space added to \topsep when environment starts a new paragraph
    (a) Is
        i. this
            A. Inception?
iii) \itemsep: space between successive items.

A more succinct way to remove all spacing is to use the option \nosep.

5.4 Buttons using Tikz (EE4)

Imports

```
\usepackage{tikz}
\usetikzlibrary{calc}
```

We can also draw a button with Tikz drawing. I'll put the code below, which is borrowed from StackOverflow.

Drawing a Button

```
\begin{tikzpicture}[
   button/.style={
   rectangle,
   minimum size=6mm,
   very thick,
   rounded corners,
   draw=red!50!black!74,
   top color=red!50!black!70,
   bottom color=white,
   }]
   \node[button] (button) {Button Text!};
   \begin{scope}[opacity=.6, transparency group]
      \draw[white,fill=white,rounded corners={2pt}] ($
          (button.north west) + (3pt,-3pt) $) rectangle ($
          ↪   (button.north east) + (-3pt,-8pt) $);
      \draw[white,fill=white,rounded corners={.5pt}] ($
          (button.north west) + (3pt,-5pt) $) rectangle ($
          ↪   (button.north east) + (-3pt,-8pt) $);
   \end{scope}
   \draw[white,fill=white,opacity=.8,rounded corners={1pt}] ($
       (button.south west) + (5pt,2pt) $) rectangle ($
       ↪   (button.south east) + (-5pt,4pt) $);
\end{tikzpicture}
```

Button Text!

6 Algorithm and Pseudocode (FF)

Imports

```
\usepackage{algorithm}
% \usepackage{algorithmicx} % automatically imported via
 ↪  algpseudocode
\usepackage[noend]{algpseudocode} % noend = no keyword "end",
 ↪  e.g. "end if"
```

Looking online for packages for writing algorithms, one immediately finds there are a variety of packages that all appear to do the same thing. Naturally, one asks: which one do I use? I especially like this answer, which I will summarize below.

1. `\algorithmic` - basic and first algorithm typesetting environment. Think of this as version 1

2. `\algorithmicx` - second and more customizable algorithm typesetting environment. Note that this package " itself doesn't define any algorithmic commands (e.g. `If`, `For`), but gives a set of macros to define such a command set". Instead, one can import predefined commands (layouts) such as `algpseudocode`.

3. `\algorithm2e` - third typesetting environment

4. `\algorithm` - float wrapper. I.e., one writes their `algorithmic` or `algorithmicx` *within* this to prevent page breaks, specify positioning, and starting line numbers. Makes it easier to read.

5. \algpseudocode - this is a layout style included in algorithmicx that mimics the style found in the algorithmic package. As previous described, this package will define the common algorithm macros in algorithmicx for us. Other layouts include algcompatible, algpascal (mimics pascal), and algc (mimics C).

LaTeXusers and myself seem to use the algorithmicx with the algpseudocode layout combined with the float wrapper algorithm. Note that if you import algpseudocode you do not need to import algorithmicx since the former imports the latter (source). The main predefined macros are \For, \While, \Repeat, \If{⟨Condition⟩} and \ElsIf and \Else, \Procedure{⟨name⟩}{⟨params⟩}, \Function{⟨name⟩}{⟨params⟩}, and \Loop. When invoking these, make sure to end with and \EndX, where X is the macro that is used. E.g., \EndFor. Lines that are just code (i.e. $x = 5 + y$) start with the macro \State.

Input an Image

```
\begin{algorithm}
\begin{algorithmic}[1]
\caption{Breadth First Search}
\Procedure{BFS}{$G=(V,E)$, $s \in V$}
  \State $X \gets \texttt{queue}(\{s\})$ \Comment{Becomes DFS
  ↪  if use stack}
  \While{$X \ne \emptyset$}
    \State $curr \gets X.\texttt{pop}()$
    \State $curr.visited = \texttt{True}$
    \State $\texttt{print}(curr)$
    \For{$v \in N(curr)$}
      \If{$! v.visited$}
        \State $X.\texttt{push}(v)$
      \EndIf
    \EndFor
  \EndWhile
\EndProcedure
\end{algorithmic}
\end{algorithm}
```

Algorithm 1 Breadth First Search

```
1: procedure BFS($G = (V, E)$, $s \in V$)
2:     $X \gets$ queue($\{s\}$)                ▷ Becomes DFS if use stack
3:     while $X \neq \emptyset$ do
4:         $curr \gets X$.pop()
5:         $curr.visited =$ True
6:         print($curr$)
7:         for $v \in N(curr)$ do
8:             if $!v.visited$ then
9:                 $X$.push($v$)
```

Note that instead of procedure, we can use Function instead. Typically, a function is synonmous with *returning* whereas Procedure is for running a set of commands.

7 Other Tricks (GG)

For increased spacings between paragraphs, we can use `\\[\defaultaddspace]` instead of the (often too large) `\\`. For between lines, we can similarly use `\\hspace{10pt}`.

While you want to include a backslash, use the command `\textbackslash`.

References