The \texttt{lmake} package*

Shengjun Pan
panshengjun@gmail.com

\[
\int_{\hat{X}_1 = \hat{A}_1}^{\hat{B}_1} \int_{\hat{X}_2 = \hat{A}_2}^{\hat{B}_2} \cdots \int_{\hat{X}_n = \hat{A}_n}^{\hat{B}_n} \left( \prod_{i=1}^{N} \phi_{i} \right) e^{-\frac{(\hat{X}_1 - \hat{\mu}_1)^2}{2\sigma_1^2}} e^{-\frac{(\hat{X}_2 - \hat{\mu}_2)^2}{2\sigma_2^2}} \cdots e^{-\frac{(\hat{X}_n - \hat{\mu}_n)^2}{2\sigma_n^2}} d\hat{X}_1 d\hat{X}_2 \cdots d\hat{X}_n
\]

1 Introduction

This package provides macros for \LaTeXe to simplify typesetting a list of phrases that fit a pattern.

\texttt{lcmd} makes a list of new commands by adding a prefix to existing commands.

\texttt{lmake} makes a list of the form \( p(i_1), p(i_2), \ldots, p(i_n) \), where \( p(.) \) stands for \textit{pattern}.

2 Usage and examples

\begin{verbatim}
\texttt{lcmd\{command\}{prefix\}{list}}
\end{verbatim}

makes a list of new commands from \texttt{command} by adding \texttt{prefix} to each item in the comma-separated \texttt{list}. If an item in the list is a macro, only its name is prefixed; the backslash is stripped away.

Examples: \footnote{This document corresponds to \texttt{lmake} v1.0, dated 2012/02/29.}

\begin{verbatim}
lcmd{\mathcal}{c}{A,X,P} defines: \cA \rightarrow A \quad \cX \rightarrow X \quad \cP \rightarrow P
lcmd{\mathbb}{c}{A,X,P} defines: \bbZ \rightarrow \mathbb{Z} \quad \bbR \rightarrow \mathbb{R} \quad \bbe \rightarrow \mathbb{E}
lcmd{\overline}{\texttt{vct}}{x,psi,phi} defines: \vctx \rightarrow \overline{x} \quad \vctpsi \rightarrow \overline{\psi} \quad \vctphi \rightarrow \overline{\phi}
\end{verbatim}

In the last example the new command for \texttt{phi} is \texttt{vctphi}, not \texttt{vct}\texttt{phi}. Notice the difference between \texttt{vctphi} and \texttt{vctpsi}.

\begin{verbatim}
lmake{[key1=]{value1},[key2=]{value2},...}
\end{verbatim}

makes a list of symbols by key-value pairs. Valid keys are described in the following table. \footnote{The examples used in this document require packages \texttt{amsmath}, \texttt{amssymb} and \texttt{graphicx}.}

\begin{verbatim}

\end{verbatim}
keys | default | description |
---|---|---|
p \i | Pattern. All occurrences of \i will be replaced by the corresponding index. |
c | Separator. |
n n | Last index. |
1 1 | First index. |
2 2 | Second index. |
d \ldots, \cdots | Dots. If the separator is comma, \ldots is used; otherwise \cdots is used. |
\dotsc, \dotsb | If amsmath is loaded, \dotsc and \dotsb are used respectively. |
ℓ | List of comma-separated symbols. |

Examples:

<table>
<thead>
<tr>
<th>what you type</th>
<th>what you see</th>
</tr>
</thead>
<tbody>
<tr>
<td>\lmake{}</td>
<td>1, 2, \ldots, n</td>
</tr>
<tr>
<td>\lmake{2=}</td>
<td>1, \ldots, n</td>
</tr>
<tr>
<td>\lmake{x_\i, c=}</td>
<td>x_1 x_2 \cdots x_n</td>
</tr>
<tr>
<td>\lmake{x_\i, N}</td>
<td>x_1, x_2, \ldots, x_N</td>
</tr>
<tr>
<td>\lmake{x_\i, ge, k}</td>
<td>x_1 \geq x_2 \geq \cdots \geq x_k</td>
</tr>
<tr>
<td>\lmake{\bar x_{\i}, \circ, 1=i, i+1}</td>
<td>\bar x_i \circ \bar x_{i+1} \circ \cdots \circ \bar x_n</td>
</tr>
<tr>
<td>\lmake{p_\i^\mu, \mu=}</td>
<td>p_1^{\mu_1} p_2^{\mu_2} \cdots p_m^{\mu_m}</td>
</tr>
<tr>
<td>\lmake{N_\i!, \alpha, \beta}</td>
<td>N_\alpha! N_\beta! \cdots N_\Gamma!</td>
</tr>
<tr>
<td>\lmake{\left(\frac{i}{i+1}\right), !}</td>
<td>\left(\frac{1}{i+1}\right) \left(\frac{2}{i+1}\right) \cdots \left(\frac{n}{i+1}\right)</td>
</tr>
<tr>
<td>\lmake{e_\i+1, c=}</td>
<td>(e_1+1) (e_2+1) \cdots (e_k+1)</td>
</tr>
<tr>
<td>\lmake{x_\i, l={1,3,5,11}}</td>
<td>x_1, x_3, x_5, x_{11}</td>
</tr>
<tr>
<td>\lmake{\rotatebox{-30}{\i}, \to, l={A,B,C,D}}</td>
<td>A \to B \to C \to D</td>
</tr>
</tbody>
</table>

Remarks:

- If ℓ is empty, the resulting list has the following form:
  p(1) c p(2) c d c p(n)
  If ℓ is not empty, the resulting list has the following form, for the example ℓ = \{x, y, z, u\}:
  p(x) c p(y) c p(z) c p(u)

- A non-empty item without = in the argument list is treated as a value. Normally the key-value pairs can appear out of order in the argument list to \lmake. For fast typing the key can be omitted. The missing key is searched, starting from the key that would follow the previous key in the table-order. Only keys without values are searched.
  For example, in \lmake{\bar x_{\i}, \circ, 1=i, i+1}, the missing key for \bar x_{\i} is p, the missing key for \circ is c since it follows p in the table. Similarly, the missing key for i+1 is 2 since it follows the previous key 1 in the table.

- A key not appearing or skipped in the argument list takes its default value, unless it’s been searched as a missing key and given a value.
  For example, in \lmake{x_\i, N}, all keys except for p and n take default values. Particularly, the key c is skipped and is treated as missing.
  Note that a skipped key does not take value empty. To force a value to be empty, use one of the following workarounds: \empty, key=, key=\empty, or key={}. 

2
3 Implementation

This section explains in detail how verb \texttt{\make}, \texttt{\cmd} and necessary internal macros are implemented.

\texttt{\L@Compare} \texttt{\L@Compare\{string1\}{string2}} compares two strings. The two arguments are fully expanded before comparison. \texttt{\if\L@Equal} is a Boolean variable for storing the result.

\begin{verbatim}
1 \newif\if\L@Equal
2 \def\L@Compare#1#2{%
3 \protect\edef\L@a{#1}\protect\edef\L@b{#2}%
4 \ifx\L@a\L@b\L@Equaltrue\else\L@Equalfalse\fi}
\end{verbatim}

\texttt{\L@FuzzyCompare} \texttt{\L@FuzzyCompare\{string1\}{string2}} tests if two strings are the same, where white spaces preceding the second argument are ignores. This allows flexible writing of the comma-separated argument to \texttt{\make} so that spaces may be inserted between an item and the previous comma.

\begin{verbatim}
5 \def\L@FuzzyCompare#1#2{%
6 \L@Compare\{#1\}{#2}\if\L@Equal\else\L@Compare\{#1\}{#2}\fi}
\end{verbatim}

\texttt{\L@SoftCompare} \texttt{\L@SoftCompare\{string1\}{string2}} similar to \texttt{\L@Compare}, but does not expand the two strings. This is usefully if either argument contains undefined macros, for example, the value to the key \texttt{p} in the argument to \texttt{\make}.

\begin{verbatim}
7 \def\L@SoftCompare#1#2{%
8 \let\L@a{#1}\let\L@b{#2}%
9 \ifx\L@a\L@b\L@Equaltrue\else\L@Equalfalse\fi}
\end{verbatim}

\texttt{\L@FuzzySoftCompare} \texttt{\L@FuzzySoftCompare\{string1\}{string2}} similar to \texttt{\L@FuzzyCompare}, but uses \texttt{\L@SoftCompare} instead of \texttt{\L@Compare}.

\begin{verbatim}
10 \def\L@FuzzySoftCompare#1#2{%
11 \L@SoftCompare\{#1\}{#2}\if\L@Equal\else\L@SoftCompare\{#1\}{#2}\fi}
\end{verbatim}

\texttt{\L@HasEqualSign} \texttt{\L@HasEqualSign\{string\}} tests if a string has an equal sign =. This is used to test if an argument to \texttt{\make} is a key-value pair or just a value. It is defined indirectly via \texttt{\L@HES}, which is a tail recursion for scanning the tokens in its argument.

\texttt{\L@Ignore} is an auxiliary macro that simply ignores its argument. The test result is stored in \texttt{\if\L@HasEqualSign}.

\begin{verbatim}
12 \def\L@Ignore#1\end{}\newif\if\L@HasEqualSign
13 \def\L@HasEqualSign#1{%
14 \L@HasEqualSignfalse\L@HES#1\end}
15 \def\L@HES#1{%
16 \ifx#1=\L@HasEqualSigntrue\let\L@Next=\L@Ignore\%
17 \else\ifx#1\end\let\L@Next=\relax\else\let\L@Next=\L@HES\fi\%
18 \fi\L@Next}
\end{verbatim}

\texttt{\L@ArName} \texttt{\L@ArName\{index\}} returns the name of a macro via a numeric index.

\begin{verbatim}
20 \def\L@ArName#1{%\ifcase#1 L@Pattern\or L@Comma\or L@Last\or%
21 L@First\or L@Second\or L@Dots\or L@List\else L@Other\fi}
\end{verbatim}
\LSet, \LGet
\LSet[index]=value;
\LGet[Index]
set and get the value of a macro by a numeric index. Numeric indices are used to locate
missing keys in the argument to \lmake.
22 \def\LSet[#1]=#2;{\global\expandafter\let\csname LArName[#1]\endcsname=#2}
23 \def\LGet[#1]{\csname LArName[#1]\endcsname}
\LLDots, \LCdots
denotes the default macro for low dots. If \amsmath is loaded before \lmake, \LLLDots is
set to \dotsc and \LCdots is set to \dotsb. Otherwise \LLLDots is set to \ldots and
\LCdots is set to \cdots.
24 \@ifpackageloaded{amsmath}
25 \{\def\LLLDots{\dotsc}\def\LCdots{\dotsb}}\def\LLLDots{\ldots}\def\LCdots{\cdots}
\LMap \LMap{function}{list}{new separator}
maps a list of comma-separated items to a new list, so that each item is transformed using the
given function, and the commas are replaced with the new separator. It is defined indirectly
indirectly via \LIterate, which is a tail recursion for scanning the comma-separated list.
\ifLStart is used to indicate if the current item is the first item, which is not preceded
by a comma, unlike the remaining items.
\LMap
27 \newif\ifLStart
28 \def\LMap#1#2#3{% 
29 \def\LSym{\empty}\def\LMFunc{#1}\def\LSep{#3}\
30 \LStarttrue\expandafter\LIterate#2,\end}
31 \def\LIterate#1,#2{\LMFunc{#1}\
32 \if#2\end\let\LNext=\relax\def\LNextarg{\empty}\
33 \else\LSep\let\LNext=\LIterate\def\LNextarg{#2}\fi\
34 \expandafter\LNext\LNextarg}
\LGetKeyValue \LGetKeyValue{key-value pair}
extracts the key and value from a string, and store them in \LKey and \LValue respectively. If
the argument has no equal sign, it is used as a value and the key is set to empty.
Otherwise, \LGetKV is called to extract the key and value.
36 \def\LGetKeyValue#1{% 
37 \def\LKey{\def\LValue{\def\LHasEqualSign{#1}}% 
38 \ifLHasEqualSign\LGetKV#1\end% 
39 \else\def\LKey{\def\LValue{#1}}% 
40 \fi\def\LGetKV#1=#2\end%
41 \def\LKey{#1}\def\LValue{#2}}
\LParse \LParse{list}
parses a list of comma-separated items, extracts each item, extract its key and value, looks
for keys p, c, n, 1, 2, d and l, and finally assigns the corresponding values to \LPattern,
\LCComma, \LLast, \LFirst, \LSecond, \LDots and \LLList respectively.
The counter \Lidx is the index of the next key yet to be assigned with an value.
43 \newcount\Lidx
If the list is not empty, then it calls a tail recursion \LPRS.
44 \def\LParse#1{\Lidx=0% 
45 \LFuzzySoftCompare{#1}{\% 
46 \ifLEqual\else\def\LExtra{\LPRS#1,\end,\fi}
An artificial item, `,end`, is added to the end of the actual list, which terminates the recursion when encountered.

47 \def\LOPRS#1,{%  
48 \LOSoftCompare(#1){\end}\ifL@Equal%  
49 \let\L@Next=\relax%  
50 \else}%  

If an item is empty, the missing key is searched and the corresponding default value is used.

51 \LOFuzzySoftCompare(#1){\ifL@Equal%  
52 \ifnum\L@idx<7%  
53 \ifcase\the\L@idx%  
54 \def\L@Default{i}%;  
55 \or\def\L@Default{,}%;  
56 \or\def\L@Default{n}%;  
57 \or\def\L@Default{1}%;  
58 \or\def\L@Default{2}%;  
59 \or\def\L@Default{3}%;  
60 \or\def\L@Default{4}%;  
61 \fi%  
62 \L@Set[\the\L@idx]=\L@Default;  
63 \advance \L@idx by 1%  
64 \fi%  

If the item is not empty, the key and value are extracted. Depending on what the key is, the value is assigned to the approximate macro.

65 \else%  
66 \LOGetKeyValue(#1){\let\LOCV=\LOValue%  
67 \LOFuzzyCompare{\LOKey}{p}\ifL@Equal\L@Set[0]=\LOCV;  
68 \else\LOFuzzyCompare{\LOKey}{c}\ifL@Equal\L@Set[1]=\LOCV;  
69 \else\LOFuzzyCompare{\LOKey}{n}\ifL@Equal\L@Set[2]=\LOCV;  
70 \else\LOFuzzyCompare{\LOKey}{1}\ifL@Equal\L@Set[3]=\LOCV;  
71 \else\LOFuzzyCompare{\LOKey}{2}\ifL@Equal\L@Set[4]=\LOCV;  
72 \else\LOFuzzyCompare{\LOKey}{d}\ifL@Equal\L@Set[5]=\LOCV;  
73 \else\LOFuzzyCompare{\LOKey}{l}\ifL@Equal\L@Set[6]=\LOCV;  
74 \fi%  
75 \L@Set[\the\L@idx]=\LOCV;  
76 \fi%  
77 \fi%  
78 \L@Set[\the\L@idx]=\LOValue%  
79 \fi%  
80 \L@GetKeyValue{#1}{\let\LOCV=\LOValue%  
81 \L@Set[\the\L@idx]=\LOCV;  
82 \L@GetKeyValue{\LOKey}{\fi%  
83 \L@Set[\the\L@idx]=\LOCV;  
84 \fi%  
85 \L@Next=\LOPRS%  
86 \fi%  
87 \fi%  
88 \fi%  

\lmake \lmake[\textit{key-value list}]  
\lmake is used twice. The first time it is used to set the default values. The second time it is used to set the values supplied by the argument list.

83 newif\ifL@FoundFirst

\L@Parse is used twice. The first time it is used to set the default values. The second time it is used to set the values supplied by the argument list.

84 newcommand{\lmake}[1][]{%
85 \begingroup%
86 \L@Parse{p=i,c=,d=,1=1,2=2,n=n,l=}%  
87 \L@Parse{#1}%  
88 \endgroup%  

The pattern is used to define the transforming function \texttt{\L@Func} by replacing all occurrences of \texttt{i} by the actual argument.

\begin{verbatim}
def\L@Func##1{\def\i{##1}\L@Get[0]}%
\end{verbatim}

If the key \texttt{d} is not given a value, its value is automatically determined by the value of the separator.

\begin{verbatim}
\def\L@Dots{\L@Ldots}\else\def\L@Dots{\L@Cdots}\
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\fi
\if\L@Equal
\L@Dots%  
\else\
\L@Func{\L@First}\L@FoundFirsttrue%
\fi%
\L@Compare{\L@List}{\empty}\ifL@Equal%
\L@FoundFirstfalse%
\L@Compare{\L@First}{\empty}\ifL@Equal%[108x713]  
\L@Map{\L@Func}{\L@List}{\L@Comma}%
\fi%
\\texttt{\L@CmdName\texttt{string}} returns the string itself if it is not a macro, otherwise returns the macro name with the backslash stripped away. This is an auxiliary function to \texttt{lcmd}.

\texttt{lcmd} \texttt{\texttt{lcmd}[command][prefix][list]} makes a list of new commands. See Section 2 for more details.
Change History

v1.0
General: Initial version ............... 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

\textbf{I}
\begin{itemize}
  \item \texttt{ifL@Equal} \hfill 1, 6, 11, 46, 48, 51, 67–74, 89, 90, 94, 96, 99, 103, 106
  \item \texttt{ifL@FoundFirst} \hfill 83, 100, 104
  \item \texttt{ifL@HasEqualSign} \hfill 13, 38
  \item \texttt{ifL@Start} \hfill 27
\end{itemize}

\textbf{L}
\begin{itemize}
  \item \texttt{L@a} \hfill 3, 4, 8, 9
  \item \texttt{L@anycmd} \hfill 114
  \item \texttt{L@ArName} \hfill 20, 22, 23
  \item \texttt{L@b} \hfill 3, 4, 8, 9
  \item \texttt{L@Cdots} \hfill 91
  \item \texttt{L@CmdName} \hfill 113, 118
  \item \texttt{L@Comma} \hfill 90, 100, 104, 107, 110
  \item \texttt{L@Compare} \hfill 1, 6, 89, 90, 94, 96, 99, 103, 106
  \item \texttt{L@CV} \hfill 66–73, 76
  \item \texttt{L@Default} \hfill 54–60, 62
  \item \texttt{L@Dots} \hfill 89, 91, 103, 104
  \item \texttt{L@Equalfalse} \hfill 4, 9
  \item \texttt{L@Equaltrue} \hfill 4, 9
  \item \texttt{L@Extra} \hfill 46
  \item \texttt{L@First} \hfill 96, 97
  \item \texttt{L@FoundFirstfalse} \hfill 95
  \item \texttt{L@FoundFirsttrue} \hfill 97, 101
  \item \texttt{L@Func} \hfill 88, 97, 101, 107, 110
  \item \texttt{L@Func} \hfill 27, 110, 119
  \item \texttt{L@Get} \hfill 88
  \item \texttt{L@GetKey} \hfill 36, 66
  \item \texttt{L@GetKV} \hfill 38, 41
  \item \texttt{L@HasEqualSign} \hfill 12, 37
  \item \texttt{L@HasEqualSignfalse} \hfill 15
  \item \texttt{L@HasEqualSigntrue} \hfill 17
  \item \texttt{L@HES} \hfill 15, 16, 18
  \item \texttt{L@Ignore} \hfill 43
  \item \texttt{L@Iterate} \hfill 30
  \item \texttt{L@Key} \hfill 37, 39, 42, 67–74
  \item \texttt{L@Last} \hfill 106, 107
  \item \texttt{L@MakeCmd} \hfill 117, 119
  \item \texttt{L@Map} \hfill 17–
  \item \texttt{L@FuzzyCompare} \hfill 5, 67–74
  \item \texttt{L@FuzzySoftCompare} \hfill 19, 33–35, 49, 80, 82
  \item \texttt{L@FuzzySoftCompare} \hfill 33–35
  \item \texttt{L@Parse} \hfill 43, 86, 87
  \item \texttt{L@PRS} \hfill 46, 47, 80
  \item \texttt{L@HES} \hfill 15, 16, 18
  \item \texttt{L@StripFirst} \hfill 114, 115
  \item \texttt{L@Sym} \hfill 29
  \item \texttt{L@Ignore} \hfill 37, 39, 42, 66
  \item \texttt{L@Value} \hfill 37, 39, 42
  \item \texttt{L@cmd} \hfill 1
  \item \texttt{L@make} \hfill 83
\end{itemize}