The `lmake` package*

Shengjun Pan
panshengjun@gmail.com

$$\int_{\hat{X}_1=\hat{A}_1}^{\hat{B}_1}\int_{\hat{X}_2=\hat{A}_2}^{\hat{B}_2}\cdots\int_{\hat{X}_n=\hat{A}_n}^{\hat{B}_n}\binom{N}{1\hat{\phi}_1,2\hat{\phi}_2,\ldots,m\hat{\phi}_m}e^{-\frac{(\hat{X}_1-\hat{\mu}_1)^2}{2\hat{\sigma}_1^2}}e^{-\frac{(\hat{X}_2-\hat{\mu}_2)^2}{2\hat{\sigma}_2^2}}\cdots e^{-\frac{(\hat{X}_n-\hat{\mu}_n)^2}{2\hat{\sigma}_n^2}}d\hat{X}_1\,d\hat{X}_2\cdots d\hat{X}_n$$

```
\lcmd{\hat}{h}{X,A,B,F,\mu,\phi,\sigma}
\lmake[\in_{\hX_\i=\hA_\i}^{\hB_\i},\empty]
\binom{N}{\lmake[\i\hphi_\i,,m]}
\lmake[e^{-\frac{(\hX_\i-\hmu_\i)^2}{2\hsigma_\i^2}},c=]
\lmake[d\hX_\i,\,]
```

1 Introduction

This package provides macros for LaTeX2e to simplify typesetting a list of phrases that fit a pattern.

`\lcmd` makes a list of new commands by adding a prefix to existing commands.

`\lmake` makes a list of the form $p(i_1), p(i_2), \ldots, p(i_n)$, where $p(.)$ stands for *pattern*.

2 Usage and examples

`\lcmd` `\lcmd{command}{prefix}{list}`

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

Examples: [1]

`\lcmd{\mathcal}{c}{A,X,P}`	defines:	`\cA` $\to \mathcal{A}$	`\cX` $\to \mathcal{X}$	`\cP` $\to \mathcal{P}$
`\lcmd{\mathbb}{c}{A,X,P}`	defines:	`\bbZ` $\to \mathbb{Z}$	`\bbR` $\to \mathbb{R}$	`\bbE` $\to \mathbb{E}$
`\lcmd{\overline}{vct}{x,psi,\phi}`	defines:	`\vctx` $\to \overline{x}$	`\vctpsi` $\to \overline{psi}$	`\vctphi` $\to \overline{\phi}$

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

`\lmake` `\lmake[[key1=]value1,[key2=]value2,...]`

makes a list of symbols by key-value pairs. Valid keys are described in the following table.

*This document corresponds to `lmake` v1.0, dated 2012/02/29.

[1] The examples used in this document require packages `amsmath, amssymb` and `graphicx`.

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:

what you type	what you see
\lmake[]	$1, 2, \ldots, n$
\lmake[2=]	$1, \ldots, n$
\lmake[x_\i,c=]	$x_1 x_2 \cdots x_n$
\lmake[x_\i,,N]	$x_1, x_2, \ldots, x_N$
\lmake[x_\i,\ge,k]	$x_1 \ge x_2 \ge \cdots \ge x_k$
\lmake[\bar x_{\i},\circ,1=i,i+1]	$\bar{x}_i \circ \bar{x}_{i+1} \circ \cdots \circ \bar{x}_n$
\lmake[p_\i^{\mu_\i},c=,m]	$p_1^{\mu_1} p_2^{\mu_2} \cdots p_m^{\mu_m}$
\lmake[N_\i!,\,,\Gamma,\alpha,\beta]	$N_\alpha! \, N_\beta! \cdots N_\Gamma!$
\lmake[\left(\frac{\i}{\i+1}\right),\!]	$\left(\frac{1}{1+1}\right)\left(\frac{2}{2+1}\right)\cdots\left(\frac{n}{n+1}\right)$
\lmake[(e_\i+1),c=,k,1,2]	$(e_1+1)(e_2+1)\cdots(e_k+1)$
\lmake[x_{\i},l={1,3,5,11}]	x_1, x_3, x_5, x_{11}
\lmake[\rotatebox{-30}{\i},\to,l={A,B,C,D}]	$\mathcal{A} \to \mathcal{B} \to \mathcal{C} \to \mathcal{D}$

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 $\ell = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{u}\}$:
  `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 able.

- 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 \lamke[x_\i,,N], all keys except for p and n take default values. Particularly, the key c is skipped and it 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={}.

3 Implementation

This section explains in details how verb\lmake, \lcmd and necessary internal macros are implemented.

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

```
\newif\ifL@Equal
\def\L@Compare#1#2{%
    \protected@edef\L@a{#1}\protected@edef\L@b{#2}%
    \ifx\L@a\L@b\L@Equaltrue\else\L@Equalfalse\fi}
```

\L@FuzzyCompare `\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 `\lmake` so that spaces may be inserted between an item and the previous comma.

```
\def\L@FuzzyCompare#1#2{%
    \L@Compare{#1}{#2}\ifL@Equal\else\L@Compare{#1}{ #2}\fi}
```

\L@SoftCompare `\L@SoftCompare{string1}{string2}`
similar to `\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 `p` in the argument to `\lmake`.

```
\def\L@SoftCompare#1#2{%
    \def\L@a{#1}\def\L@b{#2}%
    \ifx\L@a\L@b\L@Equaltrue\else\L@Equalfalse\fi}
```

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

```
\def\L@FuzzySoftCompare#1#2{%
    \L@SoftCompare{#1}{#2}\ifL@Equal\else\L@SoftCompare{#1}{ #2}\fi}
```

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

`\L@Ignore` is an auxiliary macro that simply ignores its argument. The test result is stored in `\ifL@HasEqualSign`.

```
\def\L@Ignore#1\end{}
\newif\ifL@HasEqualSign
\def\L@HasEqualSign#1{%
    \L@HasEqualSignfalse\L@HES#1\end}
\def\L@HES#1{%
    \ifx#1=\L@HasEqualSigntrue\let\L@Next=\L@Ignore%
    \else\ifx#1\end\let\L@Next=\relax\else\let\L@Next=\L@HES\fi%
    \fi\L@Next}
```

\L@ArName `\L@ArName[index]`
returns the name of a macro via a numeric index.

```
\def\L@ArName[#1]{\ifcase#1 L@Pattern\or L@Comma\or L@Last\or%
L@First\or L@Second\or L@Dots\or L@List\else L@Other\fi}
```

\L@Set,\L@Get

\L@Set[index]=value;
\L@Get[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.

```
\def\L@Set[#1]=#2;{\global\expandafter\let\csname\L@ArName[#1]\endcsname=#2}
\def\L@Get[#1]{\csname\L@ArName[#1]\endcsname}
```

\L@LDots,\L@Cdots

denotes the default macro for low dots. If amsmath is loaded before \lmake, \L@LDots is set to \dotsc and \L@Cdots is set to \dotsb. Otherwise \L@LDots is set to \ldots and \L@Cdots is set to \cdots.

```
\@ifpackageloaded{amsmath}
    {\def\L@Ldots{\dotsc}\def\L@Cdots{\dotsb}}
    {\def\L@Ldots{\ldots}\def\L@Cdots{\cdots}}
```

\L@Map

\L@Map{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 \L@Iterate, which is a tail recursion for scanning the comma-separated list.

\ifL@Start is used to indicate if the current item is the first item, which is not preceded by a comma, unlike the remaining items.

\L@Map

```
\newif\ifL@Start
\def\L@Map#1#2#3{%
    \def\L@Sym{\empty}\def\LM@Func{#1}\def\L@Sep{#3}%
     \L@Starttrue\expandafter\L@Iterate#2,\end}
\def\L@Iterate#1,#2{%
    \LM@Func{#1}%
    \ifx#2\end\let\L@Next=\relax\def\L@Nextarg{\empty}%
    \else\L@Sep\let\L@Next=\L@Iterate\def\L@Nextarg{#2}\fi%
    \expandafter\L@Next\L@Nextarg}
```

\L@GetKeyValue

\L@GetKeyValue{key-value pair}

extracts the key and value from a string, and store them in \L@Key and \L@Value respectively. If the argument has no equal sign, it is used as a value and the key is set to empty. Otherwise, \L@GetKV is called to extract the key and value.

```
\def\L@GetKeyValue#1{%
    \def\L@Key{}\def\L@Value{}\L@HasEqualSign{#1}%
    \ifL@HasEqualSign\L@GetKV#1\end%
    \else\def\L@Key{}\def\L@Value{#1}%
    \fi}
\def\L@GetKV#1=#2\end{%
    \def\L@Key{#1}\def\L@Value{#2}}
```

\L@Parse

\L@Parse{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 \L@Pattern, \L@Comma, \L@Last, \L@First, \l@Second, \L@Dots and \L@List respectively.

The counter \L@idx is the index of the next key yet to be assigned with an value.

```
\newcount\L@idx
```

If the list is not empty, then it calls a tail recursion \l@PRS.

```
\def\L@Parse#1{\L@idx=0%
    \L@FuzzySoftCompare{#1}{}%
    \ifL@Equal\else\def\L@Extra{}\L@PRS#1,\end,\fi}
```

An artificial item `,\end,` is added to the end of the actual list, which terminates the recursion when encountered.

```
\def\L@PRS#1,{%
    \L@SoftCompare{#1}{\end}\ifL@Equal%
        \let\L@Next=\relax%
    \else%
```

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

```
        \L@FuzzySoftCompare{#1}{}\ifL@Equal%
            \ifnum\L@idx<7%
                \ifcase\the\L@idx%
                    \def\L@Default{\i}%
                    \or\def\L@Default{,}%
                    \or\def\L@Default{n}%
                    \or\def\L@Default{1}%
                    \or\def\L@Default{2}%
                    \or\def\L@Default{}%
                    \or\def\L@Default{}%
                \fi%
                \L@Set[\the\L@idx]=\L@Default;%
                \advance \L@idx by 1%
            \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.

```
        \else%
            \L@GetKeyValue{#1}\let\L@CV=\L@Value%
            \L@FuzzyCompare{\L@Key}{p}\ifL@Equal\L@idx=1\L@Set[0]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{c}\ifL@Equal\L@idx=2\L@Set[1]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{n}\ifL@Equal\L@idx=3\L@Set[2]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{1}\ifL@Equal\L@idx=4\L@Set[3]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{2}\ifL@Equal\L@idx=5\L@Set[4]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{d}\ifL@Equal\L@idx=6\L@Set[5]=\L@CV;%
            \else\L@FuzzyCompare{\L@Key}{l}\ifL@Equal\L@idx=7\L@Set[6]=\L@CV;%
```

If an item is a value without a key, the missing key is searched and given the current value.

```
            \else\L@FuzzyCompare{\L@Key}{}\ifL@Equal%
                 \ifnum\L@idx<7%
                     \L@Set[\the\L@idx]=\L@CV;%
                 \fi%
                 \advance \L@idx by 1%
            \fi\fi\fi\fi\fi\fi\fi\fi\fi%
            \let\L@Next=\L@PRS%
         \fi%
         \L@Next}
```

`\lmake` `\lmake[key-value list]`

makes a list of symbols using the given key-value pairs. `\ifL@FoundFirst` is used to indicate if the first non-empty symbol is encountered; the first symbol is not preceded by a separator.

```
\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.

```
\newcommand{\lmake}[1][]{%
\begingroup%
    \L@Parse{p=\i,c={,},d=,1=1,2=2,n=n,l=}%
    \L@Parse{#1}%
```

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

```
    \def\L@Func##1{\def\i{##1}\L@Get[0]}%
```

If the key `d` is not given a value, its value is automatically determined by the value of the separator.

```
    \L@Compare{\L@Dots}{\empty}\ifL@Equal%
        \L@Compare{\L@Comma}{,}\ifL@Equal%
            \def\L@Dots{\L@Ldots}\else\def\L@Dots{\L@Cdots}%
        \fi
    \fi
```

The last step is to typeset the list of symbols. If the value to the key ℓ is missing, the typeset list starts with two symbols followed by the dots and ends with the last symbol. The symbols and dots are separated by the separator.

```
    \L@Compare{\L@List}{\empty}\ifL@Equal%
        \L@FoundFirstfalse%
         \L@Compare{\L@First}{\empty}\ifL@Equal\else%
             \L@Func{\L@First}\L@FoundFirsttrue%
         \fi%
         \L@Compare{\L@Second}{\empty}\ifL@Equal\else%
             \ifL@FoundFirst\L@Comma\fi%
             \L@Func{\L@Second}\L@FoundFirsttrue%
         \fi%
         \L@Compare{\L@Dots}{\empty}\ifL@Equal\else%
             \ifL@FoundFirst\L@Comma\fi\L@Dots%
         \fi%
         \L@Compare{\L@Last}{\empty}\ifL@Equal\else%
             \L@Comma\L@Func{\L@Last}%
         \fi%
```

If a non-empty value to the key ℓ is given, the typeset list consists of symbols from the items in the value of ℓ, transformed by the pattern function.

```
     \else%
         \L@Map{\L@Func}{\L@List}{\L@Comma}%
     \fi%
\endgroup}
```

`\L@CmdName` `\L@CmdName{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 `\lcmd`.

```
\def\L@CmdName#1{%
    \if\noexpand#1\noexpand\L@anycmd\expandafter\L@StripFirst\string#1\else#1\fi}
\def\L@StripFirst#1#2{#2}
```

`\lcmd` `\lcmd{command}{prefix}{list}`

makes a list of new commands. See Section 2 for more details.

```
\def\lcmd#1#2#3{%
    \def\L@MakeCmd##1{%
        \expandafter\def\csname #2\L@CmdName##1\endcsname{#1{##1}}}%
     \L@Map{\L@MakeCmd}{#3}{}}
```

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.

I

\ifL@Equal 1, 6, 11, 46, 48, 51, 67–74, 89, 90, 94, 96, 99, 103, 106
\ifL@FoundFirst 83, 100, 104
\ifL@HasEqualSign . 13, 38
\ifL@Start 27

L

\L@a 3, 4, 8, 9
\L@anycmd 114
\L@ArName 20, 22, 23
\L@b 3, 4, 8, 9
\L@Cdots 91
\L@CmdName 113, 118
\L@Comma 90, 100, 104, 107, 110
\L@Compare . . . 1, 6, 89, 90, 94, 96, 99, 103, 106
\L@CV 66–73, 76
\L@Default 54–60, 62
\L@Dots . . . 89, 91, 103, 104
\L@Equalfalse 4, 9
\L@Equaltrue 4, 9
\L@Extra 46
\L@First 96, 97
\L@FoundFirstfalse 95
\L@FoundFirsttrue 97, 101
\L@Func 88, 97, 101, 107, 110
\L@FuzzyCompare . . 5, 67–74
\L@FuzzySoftCompare 10, 45, 51
\L@Get 88
\L@GetKeyValue 36, 66
\L@GetKV 38, 41
\L@HasEqualSign . . . 12, 37
\L@HasEqualSignfalse . . 15
\L@HasEqualSigntrue . . . 17
\L@HES 15, 16, 18
\L@idx 43, 44, 52, 53, 62, 63, 67–73, 75, 76, 78
\L@Ignore 12, 17
\L@Iterate 30, 31, 34
\L@Key . . . 37, 39, 42, 67–74
\L@Last 106, 107
\L@Ldots 25, 26, 91
\L@LDots,\L@Cdots 24
\L@List 94, 110
\L@MakeCmd 117, 119
\L@Map 27, 110, 119
\L@Next 17–19, 33–35, 49, 80, 82
\L@Nextarg 33–35
\L@Parse 43, 86, 87
\L@PRS 46, 47, 80
\L@Second 99, 101
\L@Sep 29, 34
\L@Set 62, 67–73, 76
\L@Set,\L@Get 22
\L@SoftCompare . . 7, 11, 48
\L@Starttrue 30
\L@StripFirst 114, 115
\L@Sym 29
\L@Value 37, 39, 42, 66
\lcmd *1*, 116
\LM@Func 29, 32
\lmake *1*, 83