Package math-operator v. 1.1 Implementation

Conrad Kosowsky
March 2025
kosowsky.latex@gmail.com

Overview

The math-operator package defines control sequences for roughly one hundred and fifty math operators, including special functions, probability distributions, pure mathematical constructions, and a variant of `\overline`. The package also provides an interface for users to define new math operators similar to the amsopn package. New operators can be medium or bold weight, and they may be declared as `\mathord` or `\mathop` subformulas.

This file documents the code for the math-operator package. It is not a user guide! If you are looking for instructions on how to typeset math operators in your equations, please see `math-operator_user_guide.pdf`, which is included with the math-operator installation and is available on CTAN. The first three sections of this file deal with package setup and the commands to define new operators, and the remaining sections document the operators defined in this package. Section 4 covers several blackboard-bold letters, and Section 5 discusses categories. Section 6 deals with Jacobi elliptic functions, and Section 7 defines matrix groups and linear algebra operations. Section 8 defines the command to produce an overline. Section 9 defines a number of common probability distributions. Section 10 deals with special functions, and Section 11 covers other standard operators. Finally, Section 12 defines trigonometric functions. Version history and code index appear at the end of the document.

1 Setup

We begin with package declaration. The first 59 lines of `operator.sty` are comments.

```
60 \NeedsTeXFormat{LaTeX2e}
61 \ProvidesPackage{math-operator}[2025/03/29 Package math-operator v. 1.1]
```

Create booleans that we'll use for option processing. One boolean per category of operator.

```
62 \newif\if@operator@bb  % blackboard bold
63 \newif\if@operator@c   % category theory
64 \newif\if@operator@j   % Jacobi elliptic functions
65 \newif\if@operator@l   % linear algebra
66 \newif\if@operator@o   % overlining
67 \newif\if@operator@p   % probability distributions
68 \newif\if@operator@sf  % special functions
69 \newif\if@operator@s   % standard operators
70 \newif\if@operator@t   % trigonometry
```

Set all options to true by default.

```
\@operator@bbtrue
\@operator@ctrue
\@operator@jtrue
\@operator@ltrue
\@operator@otrue
\@operator@ptrue
\@operator@sftrue
\@operator@strue
\@operator@ttrue
```

Now declare and process options.

```
\DeclareOption{blackboard}{\@operator@bbtrue}
\DeclareOption{category}{\@operator@ctrue}
\DeclareOption{jacobi}{\@operator@jtrue}
\DeclareOption{linear}{\@operator@ltrue}
\DeclareOption{overbar}{\@operator@otrue}
\DeclareOption{probability}{\@operator@ptrue}
\DeclareOption{special}{\@operator@sftrue}
\DeclareOption{standard}{\@operator@strue}
\DeclareOption{trigonometry}{\@operator@ttrue}
```

The `no-` options prevent the package from defining certain operators.

```
\DeclareOption{no-blackboard}{\@operator@bbfalse}
\DeclareOption{no-category}{\@operator@cfalse}
\DeclareOption{no-jacobi}{\@operator@jfalse}
\DeclareOption{no-linear}{\@operator@lfalse}
\DeclareOption{no-overbar}{\@operator@ofalse}
\DeclareOption{no-probability}{\@operator@pfalse}
\DeclareOption{no-special}{\@operator@sffalse}
\DeclareOption{no-standard}{\@operator@sfalse}
\DeclareOption{no-trigonometry}{\@operator@tfalse}
\ProcessOptions*
```

A couple bookkeeping things. The count `\operatordefmode` determines the package behavior when the user calls one of the declaration commands on a control sequence that is already defined (and not an operator). When `\operatordefmode` is negative, the package overwrites the definition. Otherwise, the package behaves as follows:

- 0: Silently ignore (message written on the `log` file)
- 1 (default): issue a warning
- 2 or greater: issue an error message

We implement this behavior later in `\make@newop@cmd`.

```
\def\@operatorinfo#1{\wlog{Package operator Info: #1}}
\newcount\operatordefmode
\operatordefmode\@ne
```

Boolean that is false in general and true whenever we are typesetting a bold operator.

```
\newif\if@bfop@
\@bfop@false
```

2 Extra User-Level Commands

Some math operators contain characters other than letters, and we define control sequences to access three expressions from the operator font:

- `\@operatorhyphen` typesets a hyphen in the `\fam`
- `\@operatortw@` typesets 2 in the `\fam`
- `\@operatorm@ne` typesets -1 possibly in the `\fam`

Later in this section, we define user-level wrappers around these control sequences, and the user-level commands for squared and inverse operators format the expression as a superscript. I am assuming that the operator font contains the appropriate glyphs in the encoding slots for hyphen, `1`, and `2`, so we can define `\@operatorhyphen` and `\@operatortw@` as `\mathalpha` characters using `\mathchardef` or `\Umathchardef`.

```
\def\defaultinverse{-1}
\ifdefined\Umathchar
  \Umathchardef\@operatorhyphen=+7+0+`\-\relax
  \Umathchardef\@operatortw@=+7+0+`2\relax
```

The minus sign is tricky because it may change its location depending on the encoding. For -1, we first check if the `\fam` contains a glyph in slot `"2212`. If yes, we typeset -1 in the operator font, and if not, we forget about the operator font entirely and typeset `\defaultinverse` (which is -1 by default) with `\fam` $= -1$.

```
  \edef\@operatorm@ne{%
    \noexpand\iffontchar\textfont\fam"2212\relax
      \Umathchar+0+\fam"2212\relax
      \Umathchar+0+\fam+\number`1\relax
    \noexpand\else
      \begingroup\fam\m@ne\noexpand\defaultinverse\endgroup
    \noexpand\fi}
```

For the case where `\Umathchar` is undefined, we have to convert encoding slots for the hyphen an two numbers into hexadecimal. We use the same approach as `\set@mathchar` from the LaTeX kernel. First is the math-mode hyphen.

```
\else
  \begingroup
    \@tempcntb=\number`\-\relax
    \count@\@tempcntb
    \divide\count@ by 16\relax
    \@tempcnta\count@
    \multiply\count@ by 16\relax
    \advance\@tempcntb by -\count@
    \global\mathchardef\@operatorhyphen
      "70\hexnumber@\@tempcnta\hexnumber@\@tempcntb\relax
```

Next is 2.

```
    \@tempcntb=\number`2\relax
    \count@\@tempcntb
    \divide\count@ by 16\relax
    \@tempcnta\count@
    \multiply\count@ by 16\relax
    \advance\@tempcntb by -\count@
    \global\mathchardef\@operatortw@
      "70\hexnumber@\@tempcnta\hexnumber@\@tempcntb\relax
  \endgroup
```

For the inverse symbol, we just use `\defaultinverse`.

```
  \def\@operatorm@ne{\begingroup\fam\m@ne\defaultinverse\endgroup}
\fi
```

Now provide user-level access to those three control sequences. Each command has a starred version, which typesets the characters in bold if possible. The default, unstarred version can appear anywhere in an equation. The general idea behind both starred and unstarred versions is to first check whether `\if@bfop@` is true and then fill in values accordingly. When `\if@bfop@` is true, we can type a hyphen directly, and otherwise, we use `\@bfop@choices` or `\@operatorhyphen` depending on whether we want bold or medium weight.

```
\protected\def\operatorhyphen{\@ifstar
  \@operatorhyphen@bf\@operatorhyphen@@}
\def\@operatorhyphen@bf{%
  \if@bfop@
    -%
  \else
    \mathchoice{}{}{}{}
    {\@init@bfopfont\@bfop@choices{-}}
  \fi}
\def\@operatorhyphen@@{%
  \if@bfop@
    -%
  \else
    {\operator@font\@operatorhyphen}
  \fi}
```

Same thing for the power of 2. When `\if@bfop@` is true, we use a `\textsuperscript`, and when it is false, we expect to be in M mode and can use `^` directly.

```
\protected\def\operatorsquared{\@ifstar
  \@operatorsquared@bf\@operatorsquared@@}
\def\@operatorsquared@bf{%
  \if@bfop@
    \textsuperscript{2}%
  \else
    ^{\@init@bfopfont\@bfop@choices{2}}
  \fi}
\def\@operatorsquared@@{%
  \if@bfop@
    \textsuperscript{2}%
  \else
    ^{\operator@font\@operatortw@}
  \fi}
```

And inverse operation. This control sequence is once again more complicated because of the issue with the minus sign. As in `\operatorsquared`, we use `\textsuperscript` and type out the −1 directly when `\if@bfop@` is true, and if that boolean is false, we use `^` and `\@bfop@choices` or `\@operatorm@ne`. Unlike with `\operatorsquared`, typing out −1 is involves checks to determine whether the font contains a minus sign in slot `"2212`. If yes, we type −1 with `\Uchar`, and if not, we use `\defaultinverse`.

```
\protected\def\operatorinverse{\@ifstar
  \@operatorinverse@bf\@operatorinverse@@}
\def\@operatorinverse@bf{%
  \if@bfop@
    \textsuperscript{%
      \ifdefined\Uchar
        \iffontchar\font"2212\relax
          \Uchar"2212\relax 1%
        \else
          $\defaultinverse$%
        \fi
      \else
        $\defaultinverse$%
      \fi}%
  \else
    ^{\@init@bfopfont\@bfop@choices{%
      \ifdefined\Uchar
        \iffontchar\font"2212\relax
          \Uchar"2212\relax 1%
        \else
          $\defaultinverse$%
        \fi
      \else
        $\defaultinverse$%
      \fi}}
  \fi}
\def\@operatorinverse@@{%
  \if@bfop@
    \textsuperscript{%
      \ifdefined\Umathchar
```

```
        \iffontchar\font"2212\relax
          \Uchar"2212\relax 1%
        \else
          $\defaultinverse$%
        \fi
      \else
        $\defaultinverse$%
      \fi}%
  \else
    ^{\operator@font\@operatorm@ne}
  \fi}
```

Finally, we make a new name for pilcrow symbol because math-operator may overwrite `\P`.

```
\protected\def\pilcrow{\ifmmode
  \textparagraph\else\mathparagraph\fi}
```

3 Defining New Operators

We begin with the user-level operator declarations. Each of these commands expects the `#1` argument to be a single control sequence, and the `#2` argument should be the text of the operator. These macros all serve as wrappers around `\make@newop@cmd`, which does the work of defining the operator and expects two arguments. The first argument is the `#1` control sequence, and the second argument is code that defines `#1` as some expression involving `#2`. (We execute this code inside `\make@newop@cmd` if `#1` is a valid operator name.) In the first user-level command here, the control sequence should expand to `{\operator@font`⟨*text*⟩`}`, which simply typesets `#2` in the operator font. More complicated cases use different definitions.

```
\protected\def\DeclareMathText#1#2{\relax
  \make@newop@cmd{#1}
    {\protected\def#1{{\operator@font #2}}}}
```

Bold is more complicated for reasons discussed later in this section. The definition of `#1` in this case has three parts. First, the `\mathchoice` ensures that we are in math mode without adding anything to the current math formula. Then `\@init@bfopfont` sets up the bold operator font, and `\@bfop@choices` typesets `#2` in boldface.

```
\protected\def\DeclareBoldMathText#1#2{\relax
  \make@newop@cmd{#1}
    {\protected\def#1{\mathchoice{}{}{}{}\@init@bfopfont
      {\@bfop@choices{#2}}}}}
```

For the two commands that make a proper operator, i.e. a `\mathop` subformula, we allow for a starred version. A star means the operator is typeset with `\limits`, so any superscripts or subscripts will be directly above or below the operator. No star (the default version) means the operator is typeset with `\nolimits`, and any superscripts and subscripts will appear normally.

```
\protected\def\DeclareMathOperator{\relax\@ifstar
  \@operatorlim\@operatornolim}
```

Operator with bold text.

```
\protected\def\DeclareBoldMathOperator{\relax\@ifstar
  \@bfoperatorlim\@bfoperatornolim}
```

Next are the four commands that actually call `\make@newop@cmd` inside the previous two control sequences. They are analogous to `\DeclareMathText` and `\DeclareBoldMathText` above, except now we specify `\mathop`. We include an empty `\kern` to ensure that the subformula contains more than a single character, and this forces the baseline of the operator to line up with the rest of the equation. If TEX encounters an expression like `\mathop{`⟨*single char*⟩`}`, it will vertically center the ⟨*single char*⟩ in the middle of the equation because it thinks you're typesetting something like an integral or summation sign.

```
\def\@operatorlim#1#2{%
  \make@newop@cmd{#1}
    {\protected\def#1{\mathop{\operator@font\kern\z@#2}\limits}}}
```

Same thing with `\nolimits`.

```
\def\@operatornolim#1#2{%
  \make@newop@cmd{#1}
    {\protected\def#1{\mathop{\operator@font\kern\z@#2}\nolimits}}}
```

The bold operator works the same way as `\DeclareBoldMathText`: ensure we're in math mode (and raise an error if not), then call `\@init@bfopfont` and `\@bfop@choices` to typeset the operator.

```
\def\@bfoperatorlim#1#2{%
  \make@newop@cmd{#1}
    {\protected\def#1{\mathchoice{}{}{}{}\@init@bfopfont
      \mathop{\@bfop@choices{#2}}\limits}}}
```

Same thing with `\nolimits`.

```
\def\@bfoperatornolim#1#2{%
  \make@newop@cmd{#1}
    {\protected\def#1{\mathchoice{}{}{}{}\@init@bfopfont
      \mathop{\@bfop@choices{#2}}\nolimits}}}
```

We store the list of control sequences that code for operators in `\operator@list`. Initially the macro contains the operator control sequences from the LATEX kernel.

```
\def\operator@list{\log\lg\ln
  \lim\limsup\liminf
  \sin\arcsin\sinh
  \cos\arccos\cosh
  \tan\arctan\tanh
  \cot\coth
  \sec\csc
  \max\min\sup\inf
  \arg\ker\dim\hom\det
  \exp
  \Pr
  \gcd\deg}
```

Command to add to the list of operators.

```
\def\addto@operator@list#1{\expandafter
  \def\expandafter\operator@list\expandafter{\operator@list#1}}
```

The macro `\@ifoperator` accepts a single control sequence and checks whether it appears in `\operator@list`.

```
\def\@ifoperator#1{%
  \expandafter\in@\expandafter#1\expandafter{\operator@list}%
  \ifin@
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
```

Error and warning messages for `\make@newop@cmd`.

```
\def\OperatorBadNameError#1{%
  \PackageError{math-operator}{Invalid name\MessageBreak
    "\detokenize{#1}" for new operator}
    {I was expecting the name of the new operator\MessageBreak
    to be a single control sequence, but instead\MessageBreak
    you typed "\detokenize{#1}."\MessageBreak
    This doesn't work. To resolve this error,\MessageBreak
    make sure the first argument of the operator\MessageBreak
    declaration is a single control sequence.\MessageBreak}}
\def\OperatorCSDefWarning#1{%
  \PackageWarning{math-operator}{Control sequence\MessageBreak
    \string#1\space is already defined. Your\MessageBreak
    operator declaration was\MessageBreak ignored}}
\def\OperatorCSDefError#1{%
  \PackageError{math-operator}{Control sequence\MessageBreak
    \string#1\space already defined. Your\MessageBreak
    operator declaration was ignored}
    {You tried to define the control sequence\MessageBreak
    \string#1\space to be a new\MessageBreak
    operator even though it already has a\MessageBreak
    definition. I'm currently set to raise an\MessageBreak
    error when that happens. To resolve the\MessageBreak
    error, either pick a different control\MessageBreak
    sequence or set \string\operatordefmode\space to a \MessageBreak
    negative number to overwrite the definition.\MessageBreak}}
```

The `\make@newop@cmd` macro does the work of defining the new operator. It accepts two arguments: `#1` a control sequence and `#2` a statement defining `#1` to be something. The previous user-level `\Declare`⟨*stuff*⟩ macros provide the appropriate definitions, so we include `#2` by itself in two places. The main job of `\make@newop@cmd` is to check whether `#1` is a single control sequence and not already defined as something besides an operator.

```
\def\make@newop@cmd#1#2{%
  \expandafter\ifx\expandafter\@nnil\@gobble#1\@nnil % is #1 one token?
    \ifcat\relax\noexpand#1 % does #1 start with cs?
```

If `#1` is already an operator, we can redefine it no problem.

```
      \@ifoperator{#1}
        {\@operatorinfo{Redefining \string#1\space operator.}
          #2}  % <-- new definition happens here
```

If not, we first check whether `#1` is undefined. If yes, we define it and add it to `\operator@list`.

```
        {\ifx#1\@undefined
          \@operatorinfo{Defining new operator \string#1.}
          \addto@operator@list{#1}
          #2   % <-- new definition happens here
```

Otherwise, we consult the value of `\operatordefmode`. When this count is negative, we redefine the control sequence and turn it into an operator.

```
        \else                              % if #1 is already defined...
          \ifnum\operatordefmode<\z@       % case < 0: redefine
            \@operatorinfo{Overwriting definition of \string#1.}
            \addto@operator@list{#1}
            #2 % <-- new definition happens here
```

If `\operatordefmode` is nonnegative, we do not redefine `#1` and instead do nothing, issue a warning, or issue an error depending on the count value.

```
          \else
            \@operatorinfo{Leaving \string#1\space as is.}
            \ifcase\operatordefmode        % case 0: do nothing
            \or                            % case 1: warning
              \OperatorCSDefWarning{#1}
            \else                          % case >= 2: error
              \OperatorCSDefError{#1}
            \fi
          \fi
        \fi}
    \else
      \OperatorBadNameError{#1}
    \fi
  \else
    \OperatorBadNameError{#1}
  \fi}
```

Bold text in math mode is complicated because LaTeX doesn't provide direct access to a bold version of the operator font. My solution is to extract the operator font information from the nfss and then use that font family in boldface inside an `\hbox`. Doing so is an effective way to ensure that we are typesetting bold text, as opposed to bold math, in the operator font family. The other option would be to change the `\fam`, either with `\mathbf` or with a direct call to `\fam`, but that approach has two main drawbacks. First, `\mathbf` is intended to produce bold variable names, such as vectors in physics, not necessarily bold text, and the same may be true of other bold symbol fonts. In traditional LaTeX, `\mathbf` does produce a bolded version of the operator font, but that may not be true in general. Second, it may not be obvious whether the NFSS has even declared a bold version of the operator font as a symbol font, and I do not want to hack the NFSS to determine this when we have an easier and more reliable alternative.

To implement this approach, we need three `\font` commands (which we call `\@bfoptf`, `\@bfopsf`, and `\@bfopssf`) that switch to the bold operator font in H mode at different sizes: one size for `\textstyle` and `\displaystyle`, one size for `\scriptstyle`, and one size for `\scriptscriptstyle`. We implicitly assume that the operator font family does not change after `\begin{document}` (but it can change in the preamble), but the user may freely enlarge or shrink the text. For these three `\font` commands, we store their sizes in `\operator@tf`, `\operator@sf`, and `\operator@ssf`, and if these macros do not match the size of the current `\textfont`, `\scriptfont`, or `\scriptscriptfont` respectively, we redefine all three `\font` commands at the correct size. The idea here is that we make new `\font` commands if the surrounding text size has changed since the user's most recent bold operator. Finally, we pick the right size in the current equation by putting the operator text inside `\mathchoice`.

```
\let\operator@tf\relax
\let\operator@sf\relax
\let\operator@ssf\relax
```

The macro `\@init@bfopfont` (re)defines the font-change commands. We start by checking whether the sizes of `\@bfoptf`, `\@bfopsf`, and `\@bfopssf` match `\tf@size`, `\sf@size`, and `\ssf@size`. If yes, this command does nothing, and if not, we need to redefine the three `\font` commands to have the correct font size.

```
\def\@init@bfopfont{%
  \@tempswatrue       % reset sizes by default
  \ifx\operator@tf\tf@size
    \ifx\operator@sf\sf@size
      \ifx\operator@ssf\ssf@size
        \@tempswafalse % don't reset sizes if unnecessary
      \fi
    \fi
  \fi
```

If we do need to change the font size, we switch to the operator font inside a group and get rid of the `\escapechar`, then do fun things.

```
  \if@tempswa
    \begingroup
      \operator@font
      \escapechar\m@ne
```

The macro `\set@bf@@` accepts three arguments. The `##1` argument should is a control sequence, the `##2` argument is `\textfont`, `\scriptfont`, or `\scriptscriptfont`, and the `##3` argument is a font size (a number). This is the command that finds the operator font in the NFSS and and converts it to bold, and it defines `##1` to be the `\font` command that produces this font.

```
      \def\set@bf@@##1##2##3{%
        \edef\@tempa{\expandafter\string\the##2\fam}
        \expandafter\split@name\@tempa\@nil
        \DeclareFixedFont##1\f@encoding\f@family\bfdefault\f@shape##3}
```

Here is where we actually make the font-change commands. We call our "text font" `\@bfoptf`, our "script font" `\@bfopsf`, and our "script script font" `\@bfopssf`.

```
      \set@bf@@\@bfoptf\textfont\tf@size
      \set@bf@@\@bfopsf\scriptfont\sf@size
      \set@bf@@\@bfopssf\scriptscriptfont\ssf@size
```

Now save the sizes of the current font-change commands for the next bold operator.

```
      \global\let\operator@tf\tf@size
      \global\let\operator@sf\sf@size
      \global\let\operator@ssf\ssf@size
    \endgroup
  \fi}
```

The `\@bfop@choices` macro accepts a single argument, which should be the text of an operator to be typeset in bold. The `#1` argument goes inside an `\hbox` with a font-change command that depends on the current math style. To keep things working properly in the NFSS, we also update the internal macros that store the font information. The macro `\operator@update@font` calls `\split@name` on the current `\font`. We need to set the `\escapechar` to -1 because `\split@name` does not expect the leading backslash from the current `\font`.

```
\def\operator@update@font{\begingroup
    \escapechar\m@ne
    \expandafter\expandafter\expandafter
  \endgroup
  \expandafter\expandafter\expandafter
  \split@name\expandafter\string\the\font\@nil}
```

Now define `\@bfop@choices`. We initially set `\if@bfop@` to true. We don't have to reset it because `\@bfop@choices` always appears in a group, so this change is necessarily local.

```
\def\@bfop@choices#1{\@bfop@true
  \mathchoice{\hbox{\@bfoptf\operator@update@font#1}}
    {\hbox{\@bfoptf\operator@update@font#1}}
    {\hbox{\@bfopsf\operator@update@font#1}}
    {\hbox{\@bfopssf\operator@update@font#1}}}
```

Finally, we make the operator declaration commands preamble only. Is this necessary? Probably not, but operator declaration distinctly feels like something that should happen only in the preamble.

```
\@onlypreamble\DeclareMathText
\@onlypreamble\DeclareBoldMathText
\@onlypreamble\DeclareMathOperator
\@onlypreamble\DeclareBoldMathOperator
```

```
\@onlypreamble\@operatorlim
\@onlypreamble\@operatornolim
\@onlypreamble\@bfoperatorlim
\@onlypreamble\@bfoperatornolim
\@onlypreamble\make@newop@cmd
```

4 Blackboard Bold

Make the blackboard-bold letters. This package isn't designed to load fonts, so we're implementing each \⟨*letter*⟩ assuming the user has or will at some point request a package that defines `\mathbb`. We leave each blackboard-board letter undefined until the first time it appears in math mode. If `\mathbb` is still undefined at that point, math-operator raises a `\NoBBError`, and otherwise it defines \⟨*letter*⟩ to be `\mathbb{`⟨*letter*⟩`}`. For a traditional LaTeX approach that implements `\mathbb` as a new math alphabet, i.e. by changing the font of certain letters without changing their encoding slots, it would arguably be better to find the math family that corresponds to `\mathbb` and use `\mathchardef`. However, modern Unicode-based implementations may also change the encoding slot to something from the Letterlike Symbols or Math Alphanumeric Symbols blocks by locally setting new `\Umathcode`'s for Latin letters. Without examining the underlying code, it's impossible to know which form of `\mathbb` we're looking at, so math-operator sticks to calling `\mathbb` for maximum compatibility with other packages. If a package defines `\mathbb` to do something besides switch to blackboard bold, that will break the control sequences here.

```
\wlog{}
\if@operator@bb
  \@operatorinfo{Defining blackboard-bold letters.}
  \def\NoBBError#1{\PackageError{math-operator}
    {Missing \string\mathbb\space command}
    {It looks like you're trying to make\MessageBreak
    a blackbold-board "#1." However, I\MessageBreak
    can't define blackboard-bold letters\MessageBreak
    because I don't see a definition for\MessageBreak
    \string\mathbb, which is what I would use to\MessageBreak
    do it. To resolve this error message,\MessageBreak
    try loading a package that provides\MessageBreak
    blackboard-bold font support such as\MessageBreak
    amssymb or mathfont.\MessageBreak}}
```

To keep the code simple, we use `\@makebbchar` for the actual definitions. Here `#1` is the letter we use.

```
  \def\@makebbchar#1{%
    \@operatorinfo{Predefining
      \expandafter\string\csname #1\endcsname\space
      for use later.}
    \protected\@namedef{#1}{%
      \ifdefined\mathbb
      \@operatorinfo{Setting up \expandafter
        \string\csname#1\endcsname\space for use.}%
      \protected\global\@namedef{#1}{\mathbb{#1}}%
      \@nameuse{#1}%
    \else
      \NoBBError{#1}%
    \fi}}
\@makebbchar{N}
\@makebbchar{Z}
\@makebbchar{Q}
\@makebbchar{R}
\@makebbchar{C}
```

Defining `\H` and `\O` is more complicated because these commands already do something in text mode, and we want to preserve that definition. We take the usual approach of making them `\protected` macros that expand differently in M mode versus H or V mode.

```
\@operatorinfo{Predefining \string\H\space for use later.}
\@operatorinfo{Predefining \string\O\space for use later.}
\let\textH\H
\let\textO\O
\def\mathH{%
  \ifdefined\mathbb
    \@operatorinfo{Setting up \string\H\space for use.}%
    \protected\global\def\mathH{\mathbb{H}}%
    \mathH
  \else
    \NoBBError{H}%
  \fi}
\def\mathO{%
  \ifdefined\mathbb
    \@operatorinfo{Setting up \string\O\space for use.}%
    \protected\global\def\mathO{\mathbb{O}}%
    \mathO
  \else
    \NoBBError{O}%
  \fi}
\protected\def\O{\ifmmode\mathO\else\expandafter\textO\fi}
\protected\def\H{\ifmmode\mathH\else\expandafter\textH\fi}
```

For probability and expected value operators, we take a slightly different approach because we want these characters to show up as operators rather than `\mathord` atoms. We use `\@makebbop`, which is similar to `\@makebbchar`, except that it includes a `\mathop` specification.

```
\def\@makebbop#1{%
  \@operatorinfo{Predefining
    \expandafter\string\csname #1\endcsname\space
```

```
        for use later.}
      \protected\@namedef{#1}{%
        \ifdefined\mathbb
          \@operatorinfo{Setting up \expandafter
            \string\csname#1\endcsname\space for use.}%
          \protected\global\@namedef{#1}{\mathop{\kern\z@\mathbb{#1}}}%
          \@nameuse{#1}%
        \else
          \NoBBError{#1}%
        \fi}}
  \@makebbop{E}
  \@makebbop{P}
```

And add all these letters to `\operator@list`.

```
  \addto@operator@list{\N\Z\Q\R\C\mathO\mathH\E\P}
\else
  \@operatorinfo{Skipping blackboard-bold letters.}
\fi
```

5 Categories

A selection of categories. Serious category theorists will undoubtedly want to declare their own categories beyond these.

```
\if@operator@c
  \wlog{}
  \@operatorinfo{Defining category theory notation.}
  \DeclareBoldMathText{\Ab}{Ab}
  \DeclareBoldMathText{\Alg}{Alg}
  \DeclareBoldMathText{\Cat}{Cat}
  \DeclareBoldMathText{\CRing}{CRing}
  \DeclareBoldMathText{\Field}{Field}
  \DeclareBoldMathText{\FinGrp}{FinGrp}
  \DeclareBoldMathText{\FinVect}{FinVect}
  \DeclareBoldMathText{\Grp}{Grp}
  \DeclareBoldMathText{\Haus}{Haus}
  \DeclareBoldMathText{\Man}{Man}
  \DeclareBoldMathText{\Met}{Met}
  \DeclareBoldMathText{\Mod}{Mod}
  \DeclareBoldMathText{\Mon}{Mon}
  \DeclareBoldMathText{\Ord}{Ord}
  \DeclareBoldMathText{\Ring}{Ring}
  \DeclareBoldMathText{\Set}{Set}
  \DeclareBoldMathText{\Top}{Top}
  \DeclareBoldMathText{\Vect}{Vect}
  \DeclareMathOperator{\cocone}{cocone}
  \DeclareMathOperator{\colim}{colim}
  \DeclareMathOperator{\cone}{cone}
  \@operatorinfo{Defining new operator \string\op.}
  \addto@operator@list{\op}
  \protected\def\op{^{\operator@font op}}
\else
  \@operatorinfo{Skipping category theory notation.}
\fi
```

6 Jacobi Elliptic Functions

Pretty straightforward. The standard classes define `\sc` as a legacy (deprecated) command to access small caps, and we overwrite that definition. I do not feel bad about overwriting deprecated commands.

```
\if@operator@j
  \wlog{}
  \@operatorinfo{Defining Jacobi elliptic functions.}
  \DeclareMathOperator{\cd}{cd}
  \DeclareMathOperator{\cn}{cn}
  \DeclareMathOperator{\cs}{cs}
  \DeclareMathOperator{\dc}{dc}
  \DeclareMathOperator{\dn}{dn}
  \DeclareMathOperator{\ds}{ds}
  \DeclareMathOperator{\nc}{nc}
  \DeclareMathOperator{\nd}{nd}
  \DeclareMathOperator{\ns}{ns}
  \let\sc\@undefined
  \DeclareMathOperator{\sc}{sc}
  \DeclareMathOperator{\sd}{sd}
  \DeclareMathOperator{\sn}{sn}
\else
  \@operatorinfo{Skipping Jacobi elliptic functions.}
\fi
```

7 Linear Algebra

Some standard functions and operators from linear algebra. For the matrix groups defined here, we use `\DeclareMathText` rather than `\DeclareMathOperator` because these groups should be `\mathord` subformulas, not `\mathop`. We say `\spanop` instead of `\span` because `\span` is already defined. (It's a TeX primitive dealing with tabular entries. Please do not redefine `\span`.)

```
\if@operator@l
  \wlog{}
  \@operatorinfo{Defining operators from linear algebra.}
  \DeclareMathOperator{\adj}{adj}
  \DeclareMathOperator{\coker}{coker}
  \DeclareMathText{\GL}{GL}
  \DeclareMathOperator{\nullity}{nullity}
  \DeclareMathText{\Orthogonal}{O}
  \DeclareMathOperator{\proj}{proj}
  \DeclareMathOperator{\rank}{rank}
  \DeclareMathText{\SL}{SL}
  \DeclareMathText{\SO}{SO}
  \DeclareMathText{\SU}{SU}
  \DeclareMathOperator{\Sp}{Sp}
  \DeclareMathOperator*{\spanop}{span}
  \DeclareMathOperator{\tr}{tr}
  \@operatorinfo{Defining new operator \string\T.}
  \addto@operator@list{\T}
  \protected\def\T{^{\operator@font T}}
  \DeclareMathText{\Unitary}{U}
\else
  \@operatorinfo{Skipping operators from linear algebra.}
\fi
```

8 Overlining

In this section, we define the `\overbar` command, which produces an overline whose length lies somewhere between `\bar` and `\overline`. The user-level command is a short wrapper around the internal command `\@overb@r`. First, we define `\overbaroffset`, which will determine the placement of the overline over the argument of `\overbar`. As is standard in TeX, we store `\overbaroffset` as a count, which we identify with a scale factor because we divide it by 1000 when we use its value in `\@overb@r`.

```
\if@operator@o
  \wlog{}
  \@operatorinfo{Defining \string\overbar.}
  \newcount\overbaroffset
  \overbaroffset=800\relax
```

Version 1.0 of this package called the count `\operatorbaroffset` instead of `\overbaroffset`. We provide access to the old name for backwards compatibility.

```
  \let\operatorbaroffset\overbaroffset
```

Now for the user-level command. We start by checking whether we are in math mode. If the user follows `\overbar` with an asterisk, we use 500 as the placement value, and otherwise, we take the value of `\overbaroffset`. If the user does not specify an optional argument, we use the default value of 0.8.

```
  \protected\def\overbar{\mathchoice{}{}{}{}%
```

```
    \@ifstar
      {\@ifnextchar[%
        {\@overb@r{500}}
        {\@overb@r{500}[0.8]}}
      {\@ifnextchar[%
        {\@overb@r{\the\overbaroffset}}
        {\@overb@r{\the\overbaroffset}[0.8]}}}
```

Now we come to the macro that does the actual work of making the overline. The command `\@overb@r` accepts three arguments: `#1` is the scale argument that determines the horizontal placement of the overline; `#2` is a decimal that determines the size of the overline; and `#3` is the subformula to be overlined. Note that `#1` is only specified internally, and the user controls `#1` indirectly through `\overbaroffset`. Just to be safe, we put the contents of `\@overb@r` inside a group since we're making liberal use of scratch registers. The general idea is that inside a `\mathchoice`, we put the subformula in an `\hbox` in the correct style and then manually position the overline using the box dimensions and the `#1` and `#2` arguments.

```
  \def\@overb@r#1[#2]#3{\begingroup
    \mathchoice
      {\setbox\@tempboxa\hbox{$\displaystyle#3\m@th$}
        \@tempdima\wd\@tempboxa
        \@tempdimb\ht\@tempboxa
        \@tempdimc\dp\@tempboxa
```

Inside another `\hbox`, we typeset `\@tempboxa` and manually convert it into an `\rlap`. Then we add `\hskip`, the overline, and an `\hfil`. In total, this `\hbox` will have the same width as the original subformula. We do all of this inside a second `\hbox` because we want TEX to treat everything as a single subformula and also to prevent extra interatom space in the unlikely event that `\Umathordordspacing` is nonzero. (Please do not set `\Umathordordspacing` to something nonzero.)

```
        \hbox to \@tempdima{\unhbox\@tempboxa\hskip-\@tempdima
          \hskip\dimexpr(\@tempdima-#2\@tempdima)*#1/1000\relax
```

Now make a math expression that contains just an overline of the correct width above a zero-width `\vrule`, which ensures that the overline occurs at the correct height.

```
          $\overline{%
            \vrule width \z@ height \@tempdimb depth \@tempdimc
            \hskip\dimexpr#2\@tempdima\relax}\m@th$%
          \hfil}}
```

Same thing with `\textstyle`.

```
      {\setbox\@tempboxa\hbox{$\textstyle#3\m@th$}
        \@tempdima\wd\@tempboxa
        \@tempdimb\ht\@tempboxa
        \@tempdimc\dp\@tempboxa
        \hbox to \@tempdima{\unhbox\@tempboxa\hskip-\@tempdima
          \hskip\dimexpr(\@tempdima-#2\@tempdima)*#1/1000\relax
          $\overline{%
            \vrule width \z@ height \@tempdimb depth \@tempdimc
            \hskip\dimexpr#2\@tempdima\relax}\m@th$%
          \hfil}}
```

And `\scriptstyle`.

```
    {\setbox\@tempboxa\hbox{$\scriptstyle#3\m@th$}
      \@tempdima\wd\@tempboxa
      \@tempdimb\ht\@tempboxa
      \@tempdimc\dp\@tempboxa
      \hbox to \@tempdima{\unhbox\@tempboxa\hskip-\@tempdima
        \hskip\dimexpr(\@tempdima-#2\@tempdima)*#1/1000\relax
        $\overline{%
          \vrule width \z@ height \@tempdimb depth \@tempdimc
          \hskip\dimexpr#2\@tempdima\relax}\m@th$%
        \hfil}}
```

And finally `\scriptscriptstyle`.

```
    {\setbox\@tempboxa\hbox{$\scriptscriptstyle#3\m@th$}
      \@tempdima\wd\@tempboxa
      \@tempdimb\ht\@tempboxa
      \@tempdimc\dp\@tempboxa
      \hbox to \@tempdima{\unhbox\@tempboxa\hskip-\@tempdima
        \hskip\dimexpr(\@tempdima-#2\@tempdima)*#1/1000\relax
        $\overline{%
          \vrule width \z@ height \@tempdimb depth \@tempdimc
          \hskip\dimexpr#2\@tempdima\relax}\m@th$%
        \hfil}}
  \endgroup}
\else
  \@operatorinfo{Skipping \string\overbar.}
\fi
```

9 Probability Distributions

A selection of common probability distributions. We say `\Betaop` and `\Gammaop` instead of `\Beta` and `\Gamma` because they are or may be defined to be capital Greek letters.

```
\if@operator@p
  \wlog{}
  \@operatorinfo{Defining probability distributions.}
  \DeclareMathOperator{\Bernoulli}{Bernoulli}
  \DeclareMathOperator{\Betaop}{Beta}
  \DeclareMathOperator{\Binomial}{Binomial}
  \DeclareMathOperator{\Boltzmann}{Boltzmann}
  \DeclareMathOperator{\Burr}{Burr}
  \DeclareMathOperator{\Categorical}{Categorical}
```

```
\DeclareMathOperator{\Cauchy}{Cauchy}
\DeclareMathOperator{\chiop}{\chi}
\protected\def\ChiSq{\chiop^{\operator@font\@operatortw@}}
\DeclareMathOperator{\Dagum}{Dagum}
\DeclareMathOperator{\Exponential}{Exponential}
\DeclareMathOperator{\Erlang}{Erlang}
\DeclareMathOperator{\Gammaop}{Gamma}
\DeclareMathOperator{\Gompertz}{Gompertz}
\protected\def\InvChiSq{%
  \mathop{\operator@font
    Inv\@operatorhyphen\chiop}%
    \nolimits^{\operator@font\@operatortw@}}
\DeclareMathOperator{\InvGamma}
  {Inv\@operatorhyphen Gamma}
\DeclareMathOperator{\Kolmogorov}
  {Kolmogorov}
\DeclareMathOperator{\LogLogistic}
  {Log\@operatorhyphen Logistic}
\DeclareMathOperator{\LogNormal}
  {Log\@operatorhyphen Normal}
\DeclareMathOperator{\Logistic}{Logistic}
\DeclareMathOperator{\Lomax}{Lomax}
\DeclareMathOperator{\MaxwellBoltzmann}
  {Maxwell\@operatorhyphen Boltzmann}
\DeclareMathOperator{\Multinomial}{Multinomial}
\DeclareMathOperator{\NegBinomial}
  {Neg\@operatorhyphen Binomial}
```

We can't use `\N` for calligraphic $\mathcal{N}$ because we're already using it for natural numbers. Alas, we will settle for `\Normal`. This one is also more complicated because of the `*`-ed version of the command.

```
\@operatorinfo{Defining new operator \string\Normal.}
\addto@operator@list{\Normal}
\def\operator@N{\mathop{\kern\z@\mathcal{N}}\nolimits}
\def\operator@normal{\mathop{\operator@font Normal}\nolimits}
\protected\def\Normal{%
  \mathchoice{}{}{}{}
  \@ifstar\operator@normal\operator@N}
\DeclareMathOperator{\Pareto}{Pareto}
\DeclareMathOperator{\Poisson}{Poisson}
\DeclareMathOperator{\Weibull}{Weibull}
\DeclareMathOperator{\Zipf}{Zipf}
\else
  \@operatorinfo{Skipping probability distributions.}
\fi
```

10 Special Functions

Some common special functions.

```
\if@operator@sf
  \wlog{}
  \@operatorinfo{Defining special functions.}
  \DeclareMathOperator{\Ai}{Ai}
  \DeclareMathOperator{\Bi}{Bi}
  \DeclareMathOperator{\Ci}{Ci}
  \DeclareMathOperator{\ci}{ci}
  \DeclareMathOperator{\Chi}{Chi}
  \DeclareMathOperator{\Ei}{Ei}
  \DeclareMathOperator{\erf}{erf}
  \protected\def\erfinv{\erf^{\operator@font\@operatorm@ne}}
  \DeclareMathOperator{\erfc}{erfc}
  \protected\def\erfcinv{\erfc^{\operator@font\@operatorm@ne}}
  \DeclareMathOperator{\Li}{Li}
  \DeclareMathOperator{\li}{li}
  \DeclareMathOperator{\Log}{Log}
  \DeclareMathOperator{\sgn}{sgn}
  \DeclareMathOperator{\Si}{Si}
  \DeclareMathOperator{\si}{si}
  \DeclareMathOperator{\Shi}{Shi}
```

Inverse error function.

```
\else
  \@operatorinfo{Skipping special functions.}
\fi
```

11 Standard Operators

Some other standard (or standardish) functions and operations. Much more pure mathy than the special functions from the previous section. We say `\divop` instead of `\div` because `\div` is already defined.

```
\if@operator@s
  \wlog{}
  \@operatorinfo{Defining standard operators.}
  \DeclareMathOperator*{\argmax}{arg\,max}
  \DeclareMathOperator*{\argmin}{arg\,min}
  \DeclareMathOperator{\Aut}{Aut}
  \@operatorinfo{Defining new operator \string\c.}
  \addto@operator@list{\mathc}
  \let\textc\c
  \def\mathc{^{\operator@font c}}
  \protected\def\c{\ifmmode\mathc\else\expandafter\textc\fi}
  \DeclareMathOperator{\cf}{cf}
  \DeclareMathOperator{\cl}{cl}
  \DeclareMathOperator{\conv}{conv}
  \DeclareMathOperator{\corr}{corr}
  \DeclareMathOperator{\cov}{cov}
  \DeclareMathOperator{\curl}{curl}
  \DeclareMathOperator{\divop}{div}
  \DeclareMathOperator{\grad}{grad}
  \DeclareMathOperator{\Hess}{\mathcal{H}}
  \DeclareMathOperator{\Hom}{Hom}
  \DeclareMathOperator{\id}{id}
  \DeclareMathOperator{\img}{img}
  \DeclareMathOperator{\Info}{\mathcal{I}}
  \DeclareMathOperator{\interior}{int}
  \DeclareMathOperator*{\lcm}{lcm}
  \DeclareMathOperator{\Proj}{Proj}
  \DeclareMathOperator{\Res}{Res}
  \DeclareMathOperator{\Spec}{Spec}
  \DeclareMathOperator{\supp}{supp}
  \addto@operator@list{\varIm\varRe\Im\Re}
  \@operatorinfo{Defining \string\varIm\space operator from \string\Im.}
  \@operatorinfo{Defining \string\varRe\space operator from \string\Re.}
  \let\varIm\Im
  \let\varRe\Re
  \DeclareMathOperator{\Im}{Im}
  \DeclareMathOperator{\Re}{Re}
  \DeclareMathOperator{\Var}{Var}
\else
  \@operatorinfo{Skipping standard operators.}
\fi
```

12 Trigonometry

Finally, time for some trigonometry. We start by defining hyperbolic cosecant and secant.

```
\if@operator@t
  \wlog{}
  \@operatorinfo{Defining trigonometric functions.}
  \DeclareMathOperator{\csch}{csch}
  \DeclareMathOperator{\sech}{sech}
```

Round out the “arc” versions of standard trigonometric functions, then provide “arc” and “ar” versions of hyperbolic trigonometric functions.

```
  \DeclareMathOperator{\arccsc}{arccsc}
  \DeclareMathOperator{\arcsec}{arcsec}
  \DeclareMathOperator{\arccot}{arccot}
  \DeclareMathOperator{\arcsinh}{arcsinh}
  \DeclareMathOperator{\arccosh}{arccosh}
  \DeclareMathOperator{\arctanh}{arctanh}
  \DeclareMathOperator{\arccsch}{arccsch}
  \DeclareMathOperator{\arcsech}{arcsech}
  \DeclareMathOperator{\arccoth}{arccoth}
  \DeclareMathOperator{\arsinh}{arsinh}
  \DeclareMathOperator{\arcosh}{arcosh}
  \DeclareMathOperator{\artanh}{artanh}
  \DeclareMathOperator{\arcsch}{arcsch}
  \DeclareMathOperator{\arsech}{arsech}
  \DeclareMathOperator{\arcoth}{arcoth}
```

Inverse functions with -1 superscript.

```
  \protected\def\sininv{\sin^{\operator@font\@operatorm@ne}}
  \protected\def\cosinv{\cos^{\operator@font\@operatorm@ne}}
  \protected\def\taninv{\tan^{\operator@font\@operatorm@ne}}
  \protected\def\cscinv{\csc^{\operator@font\@operatorm@ne}}
  \protected\def\secinv{\sec^{\operator@font\@operatorm@ne}}
  \protected\def\cotinv{\cot^{\operator@font\@operatorm@ne}}
```

And for hyperbolic trigonometric functions.

```
  \protected\def\sinhinv{\sinh^{\operator@font\@operatorm@ne}}
  \protected\def\coshinv{\cosh^{\operator@font\@operatorm@ne}}
  \protected\def\tanhinv{\tanh^{\operator@font\@operatorm@ne}}
  \protected\def\cschinv{\csch^{\operator@font\@operatorm@ne}}
  \protected\def\sechinv{\sech^{\operator@font\@operatorm@ne}}
  \protected\def\cothinv{\coth^{\operator@font\@operatorm@ne}}
\else
  \@operatorinfo{Skipping trigonometric functions.}
\fi
\wlog{}
```

Done!

Version History

New features and updates with each version. Listed in no particular order.

1.0 . . . . February 2025
—initial release

1.1 . . . . March 2025
—big fix in package declaration
—bug fix for operators with superscripts
—added `\operatorsquared` and `\operatorinverse`
—added `\Hess` and `\Info`
—changed `\operatorbaroffset` to `\overbaroffset`

Index