The mathtools package*

Morten Høgholm, Lars Madsen and the \LaTeX3 project

2024/03/11

Abstract

The mathtools package is an extension package to amsmath. There are two things on mathtools’ agenda: (1) correct various bugs/deficiencies in amsmath until these are fixed by the \AMS and (2) provide useful tools for mathematical typesetting, be it a small macro for typesetting a prescript or an underbracket, or entirely new display math constructs such as a \texttt{multlined} environment.

Contents

1 Introduction 2

2 Package loading 2
  2.1 Special \texttt{mathtools} options ........................................ 3

3 Tools for mathematical typesetting 4
  3.1 Fine-tuning mathematical layout ........................................... 4
    3.1.1 A complement to \texttt{\textbackslash smash}, \texttt{\textbackslash llap}, and \texttt{\textbackslash rlap} .................. 4
    3.1.2 Forcing a cramped style ............................................. 5
    3.1.3 Smashing an operator ................................................ 7
    3.1.4 Adjusting limits of operators ...................................... 8
    3.1.5 Swapping space above \AMS display math environments ....... 8
  3.2 Controlling tags ............................................................ 9
    3.2.1 The appearance of tags ................................................ 9
    3.2.2 Showing only referenced tags ...................................... 10
  3.3 Extensible symbols ....................................................... 12
    3.3.1 Arrow-like symbols .................................................. 12
    3.3.2 Braces and brackets .................................................. 14
  3.4 New mathematical building blocks ....................................... 15
    3.4.1 Matrices ............................................................... 16
    3.4.2 The \texttt{multlined} environment ................................ 17
    3.4.3 More cases-like environments ..................................... 19
    3.4.4 Emulating indented lines in alignments ......................... 20
    3.4.5 Boxing a single line in an alignment ............................ 20
    3.4.6 Adding arrows between lines in an alignment ................. 22
    3.4.7 Centered \textbackslash vdots ..................................... 22
  3.5 Intertext and short intertext ........................................... 24
  3.6 Paired delimiters ........................................................ 25

*This file has version number v1.30, last revised 2024/03/11.
1 Introduction

Although amsmath provides many handy tools for mathematical typesetting, it is nonetheless a static package. This is not a bad thing, because what it does, it mostly does quite well and having a stable math typesetting package is “a good thing.” However, amsmath does not fulfill all the needs of the mathematical part of the \LaTeX community, resulting in many authors writing small snippets of code for tweaking the mathematical layout. Some of these snippets have also been posted to newsgroups and mailing lists over the years, although more often than not without being released as stand-alone packages.

The mathtools package is exactly what its name implies: tools for mathematical typesetting. It is a collection of many of these often needed small tweaks—with some big tweaks added as well. It can only do so by having harvesting newsgroups for code and/or you writing the maintainers with wishes for code to be included, so if you have any good macros or just macros that help you when writing mathematics, then don’t hesitate to report them.

As of 2020, mathtools (and empheq) is now hosted at the \LaTeX3 team GitHub at

https://github.com/latex3/mathtools

So if you have any issue, feel free to register an issue there.

Update 2013: We now make \( and \[ ] robust (can be disabled via nonrobust package option).

2 Package loading

The mathtools package requires amsmath but is able to pass options to it as well. Thus a line like

\usepackage[fleqn,tbtags]{mathtools}
is equivalent to
\usepackage[fleqn,tbtags]{amsmath}
\usepackage{mathtools}

2.1 Special mathtools options

\textcolor{red}{\textbf{fixamsmath donotfixamsmathbugs}}

The option \texttt{fixamsmath} (default) fixes two bugs in \texttt{amsmath}.\footnote{See the online \LaTeX{} bugs database \url{http://www.latex-project.org/cgi-bin/latexbugs2html} under \texttt{AMS-\LaTeX} problem reports 3591 and 3614.} Should you for some reason not want to fix these bugs then just add the option \texttt{donotfixamsmathbugs} (if you can do it without typos). The reason for this extremely long name is that I really don’t see why you wouldn’t want these bugs to be fixed, so I’ve made it slightly difficult not to fix them.

\textcolor{red}{\textbf{allows}paces \textbf{disallows}paces}

Sometimes \texttt{amsmath} gives you nasty surprises, as here where things look seemingly innocent:
\[
\begin{gathered}
[t] = 100 \\
[v] = 200
\end{gathered}
\]
Without \texttt{mathtools} this will result in this output:
\[
= 100 \\
[v] = 200
\]
Yes, the \texttt{[t]} has been gobbled without warning! Note, with modern \texttt{amsmath} \texttt{\begin{gathered}\{x\}} will eat \texttt{[x]} and \textit{issue a warning unless} \texttt{x} is \texttt{t}, \texttt{b} or \texttt{c}.\footnote{Older \texttt{amsmath} will silently gobble the argument.} This is hardly what you’\texttt{d expect as an end user, as the desired output was probably something like this instead:}
\[
[t] = 100 \\
[v] = 200
\]
With the option \texttt{disallowspaces} (default) \texttt{mathtools} disallows spaces in front of optional arguments where it could possibly cause problems just as \texttt{amsmath} does with \texttt{\\} inside the display environments. This includes the environments \texttt{gathered} (and also those shown in §4.5 on page 35), \texttt{aligned}, \texttt{multlined}, and the extended \texttt{matrix} environments (§3.4.1 on page 16). If you however want to preserve the more dangerous standard optional spaces, simply choose the option \texttt{allowspace}s.
As of Summer 2022, we've had to rename several \ldots colon\ldots symbol macros in order to have a consistent naming scheme across different packages.\footnote{For example with unicode-math.} If needed, this package option brings back the original \ldots colon\ldots definitions. See page 31 for more information.

3 Tools for mathematical typesetting

\mathtoolsset{\langle key val list \rangle}

Many of the tools shown in this manual can be turned on and off by setting a switch to either true or false. In all cases it is done with the command \mathtoolsset. A typical use could be something like

\begin{verbatim}
\mathtoolsset{
  showonlyrefs,  
  mathic \% or mathic = true
}
\end{verbatim}

More information on the keys later on.

3.1 Fine-tuning mathematical layout

Sometimes you need to tweak the layout of formulas a little to get the best result and this part of the manual describes the various macros mathtools provides for this.

3.1.1 A complement to \texttt{\textbackslash smash}, \texttt{\textbackslash llap}, and \texttt{\textbackslash rlap}

\begin{verbatim}
\mathllap{\langle mathstyle \rangle}{\langle math \rangle}
\mathrlap{\langle mathstyle \rangle}{\langle math \rangle}
\mathclap{\langle text \rangle}
\mathmakebox{\langle width \rangle}{\langle pos \rangle}{\langle math \rangle}
\end{verbatim}

In [1], Alexander R. Perlis describes some simple yet useful macros for use in math displays. For example the display

\begin{verbatim}
X = \sum_{1 \le i \le j \le n} X_{ij}
\end{verbatim}

contains a lot of excessive white space. The idea that comes to mind is to fake the width of the subscript. The command \texttt{\mathclap} puts its argument in a zero width box and centers it, so it could possibly be of use here.

\begin{verbatim}
X = \sum_{\mathclap{1 \le i \le j \le n}} X_{ij}
\end{verbatim}
\[ X = \sum_{1 \leq i \leq j \leq n} X_{ij} \]

For an in-depth discussion of these macros I find it better to read the article; an online version can be found at


Note that the definitions shown in the article do not exactly match the definitions in mathtools. Besides providing an optional argument for specifying the desired math style, these versions also work around a most unfortunate \LaTeX{} “feature.” The \texttt{\smash} macro is fixed too.

### 3.1.2 Forcing a cramped style

\begin{verbatim}
\texttt{\cramped\[}\langle\text{mathstyle}\rangle\{\text{math}\}\texttt{\]}\end{verbatim}

Let’s look at another example where we have used \texttt{\mathclap}:

\begin{verbatim}
\begin{equation}\label{eq:mathclap}
\sum_{\mathclap{a^2<b^2<c}} \quad \sum_{a^2<b^2<c}
\end{equation}
\end{verbatim}

Do you see the difference? Maybe if I zoomed in a bit:

\[ \sum_{a^2<b^2<c} \quad \sum_{a^2<b^2<c} \]

\begin{equation}
(1)
\end{equation}

Notice how the limit of the right summation sign is typeset in a more compact style than the left. It is because \LaTeX{} sets the limits of operators in a cramped style. For each of \LaTeX{}’s four math styles (\texttt{\displaystyle}, \texttt{\textstyle}, \texttt{\scriptstyle}, and \texttt{\scriptscriptstyle}), there also exists a cramped style that doesn’t raise exponents as much. Besides in the limits of operators, \LaTeX{} also automatically uses these cramped styles in radicals such as \texttt{\sqrt} and in the denominators of fractions, but unfortunately there are no primitive commands that allow you to detect crampedness or switch to it.

\texttt{mathtools} offers the command \texttt{\cramped} which forces a cramped style in normal un-cramped math. Additionally you can choose which of the four styles you want it in as well by specifying it as the optional argument:

\begin{verbatim}
[\cramped{\langle\text{mathstyle}\rangle}\{\text{math}\}[\cramped{\scriptstyle}\{\langle\text{mathstyle}\rangle\}\{\text{math}\}} \leftarrow x^{-2} \quad \quad \leftarrow x^{-2}
\end{verbatim}

\begin{verbatim}
4The faulty reboxing procedure.
\end{verbatim}
You may be surprised how often the cramped style can be beneficial to your output. Take a look at this example:

\begin{quote}
The 2005 Euro\TeX{} conference is held in Abbaye des Prémontrés, France, marking the 16th $(2^2)^2$ anniversary of both Dante and GUTenberg (the German and French \TeX{} users group resp.).
\end{quote}

The 2005 Euro\TeX{} conference is held in Abbaye des Prémontrés, France, marking the 16th $(2^2)^2$ anniversary of both Dante and GUTenberg (the German and French \TeX{} users group resp.).

Typesetting on a grid is generally considered quite desirable, but as the second line of the example shows, the exponents of 2 cause the line to be too tall for the normal value of \baselineskip, so \TeX{} inserts a \lineskip (normal value is 1.0pt). In order to circumvent the problem, we can force a cramped style so that the exponents aren’t raised as much:

\begin{quote}
The 2005 Euro\TeX{} ... 16th $(\cramped{2^{2^2}})$ ...
\end{quote}

The commands \crampedllap, \crampedclap, and \crampedrlap are identical to the three \mathXlap commands described earlier except the argument is typeset in cramped style. You need this in order to typeset (1) correctly while still faking the width of the limit.

\begin{equation*}
\sum_{\crampedclap{a^2<b^2<c}} \tag{\ref{eq:mathclap}*}
\end{equation*}

$\sum_{a^2<b^2<c}$ (1*)

Of course you could just type

\begin{verbatim}
\sum_{\mathclap{\cramped{a^2<b^2<c}}} \tag{\ref{eq:mathclap}*}
\end{verbatim}

but it has one major disadvantage: In order for \mathXlap and \cramped to get the right size, \TeX{} has to process them four times, meaning that nesting them as shown above will cause \TeX{} to typeset 4$^2$ instances before choosing the right one. In this situation however, we will of course need the same style for both commands so it makes
sense to combine the commands in one, thus letting \TeX make the choice only once rather than twice.

\begin{crampedsubarray}{(|col|) (contents) \end{crampedsubarray}
\crampedsubstack{|(lines separated by \)|}

If we go back to (1) and apply \substack, you’ll notice that the cramped style, the sum would normally apply, is now gone:

\[
\sum_{a^2<b^2<c} \quad \sum_{\crampedsubstack{a^2<b^2<c}}
\]

We therefore provide a cramped version of \substack.\footnote{\substack is internally implemented via the subarray-env, so our cramped version of \substack is implemented via a cramped version of this env.}

\[
\sum_{\crampedsubstack{a^2<b^2<c}} \quad \sum_{a^2<b^2<c}
\]

Note: We may need to add a similar hook into multlined.

3.1.3 \textbf{Smashing an operator}

\smashoperator[|pos|]{|operator with limits|}

Above we showed how to get \LaTeX to ignore the width of the subscript of an operator. However this approach takes a lot of extra typing, especially if you have a wide superscript, meaning you have to put in \crampedclap in both sub- and superscript. To make things easier, mathtools provides a \smashoperator command, which simply ignores the width of the sub- and superscript. It also takes an optional argument, 1, r, or lr (default, rl is an alias for lr), denoting which side of the operator should be ignored (smashed).

\[
V = \sum_{1\le i\le j\le n}^{\infty} V_{ij} \quad \text{quad} \quad X = \smashoperator[1]{\sum_{1\le i\le j\le n}}^{3456} X_{ij} \quad \text{quad} \quad Y = \smashoperator[r]{{\sum\limits}_{1\le i\le j\le n}} Y_{ij} \quad \text{quad} \quad Z = \smashoperator[l]\mathop{T}_{1\le i\le j\le n} Z_{ij}
\]

V \substack is internally implemented via the subarray-env, so our cramped version of \substack is implemented via a cramped version of this env.
\[
V = \sum_{1 \leq i \leq j \leq n} V_{ij} \quad X = \sum_{1 \leq i \leq j \leq n} X_{ij} \quad Y = \sum_{1 \leq i \leq j \leq n} Y_{ij} \quad Z = T_{1 \leq i \leq j \leq n} Z_{ij}
\]

Note that \texttt{smashoperator} always sets its argument in display style and with limits even if you have used the \texttt{nosumlimits} option of \texttt{amsmath}. If you wish, you can use shorthands for _ and ^ such as \texttt{sb} and \texttt{sp}.

### 3.1.4 Adjusting limits of operators

\texttt{\adjustlimits{\langle\text{operator}1\rangle}_{\langle\text{limit}1\rangle}^{\langle\text{operator}2\rangle}_{\langle\text{limit}2\rangle}}

When typesetting two consecutive operators with limits one often wishes the limits of the operators were better aligned. Look closely at these examples:

\[
\text{a)} \lim_{n \to \infty} \max_{p \geq n} \quad \text{b)} \lim_{n \to \infty} \max_{p^2 \geq n} \quad \text{c)} \lim_{n \to \infty} \sup_{p^2 \geq nK} \quad \text{d)} \limsup_{n \to \infty} \max_{p \geq n}
\]

a) looks okay, but b) is not quite as good because the second limit (\(p^2 \geq n\)) is significantly taller than the first (\(n \to \infty\)). With c) things begin to look really bad, because the second operator has a descender while the first doesn’t, and finally we have d) which looks just as bad as c). The command \texttt{\adjustlimits} is useful in these cases, as you can just put it in front of these consecutive operators and it’ll make the limits line up.

\[
\text{a)} \lim \max_{n \to \infty \ p \geq n} \quad \text{b)} \lim \max_{n \to \infty \ p^2 \geq n} \quad \text{c)} \lim \sup_{n \to \infty \ p^2 \geq nK} \quad \text{d)} \limsup_{n \to \infty \ p \geq n}
\]

The use of \texttt{sb} instead of _ is allowed.

### 3.1.5 Swapping space above \texttt{AMS} display math environments

One feature that the plain old \texttt{equation} environment has that the \texttt{AMS} environments do not (because of technical reasons) is the feature of using less space above the equation if the situation presents itself. The \texttt{AMS} environments cannot do this, but one can manually, using

\texttt{\SwapAboveDisplaySkip}

as the very first content within an \texttt{AMS} display math environment. It will then issue an \texttt{\abovedisplayshortskip} instead of the normal \texttt{\abovedisplayskip}.
Note it will not work with the equation or multiline environments. Here is an example of the effect

\nindent\rule{\textwidth}{1pt}
\begin{align*}
A & = B \\
\end{align*}
\nindent\rule{\textwidth}{1pt}
\begin{align*}
\text{SwapAboveDisplaySkip}
A & = B \\
\end{align*}

\[
A = B
\]

3.2 Controlling tags

In this section various tools for altering the appearance of tags are shown. All of the tools here can be used at any point in the document but they should probably affect the whole document, so the preamble is the best place to issue them.

3.2.1 The appearance of tags

\begin{verbatim}
\newtagform{〈name〉}{〈inner_format〉}{〈left〉}{〈right〉}
\renewtagform{〈name〉}{〈inner_format〉}{〈left〉}{〈right〉}
\usetagform{〈name〉}
\end{verbatim}

Altering the layout of equation numbers in amsmath is not very user friendly (it involves a macro with three @’s in its name), so mathtools provides an interface somewhat reminiscent of the page style concept. This way you can define several different tag forms and then choose the one you prefer.

As an example let’s try to define a tag form which puts the equation number in square brackets. First we define a brand new tag form:

\begin{verbatim}
\newtagform{brackets}{[}{]}{}
\end{verbatim}

Then we activate it:

\begin{verbatim}
\usetagform{brackets}
\end{verbatim}

The result is then

\[
E \neq mc^3 [2]
\]

Similarly you could define a second version of the brackets that prints the equation number in bold face instead

\begin{verbatim}
\newtagform{brackets2}\{\textbf\}{[]}
\usetagform{brackets2}
\begin{equation}
E \neq mc^3
\end{equation}
\end{verbatim}
When you reference an equation with \eqref, the tag form in effect at the time of refer-
encing controls the formatting, so be careful if you use different tag forms throughout
your document.

If you want to renew a tag form, then use the command \renewtagform. Should
you want to return to the standard setting then choose

\usetagform{default}

Caveat regarding ntheorem: If you like to change the appearance of the tags and you
are also using the ntheorem package, then please postpone the change of appearance
until after loading ntheorem. (In order to do its thing, ntheorem has to mess with the
tags...)

3.2.2 Showing only referenced tags

\showonlyrefs = true|false
\showmanualtags = true|false
\refeq{(label)}

First of, one should always use \eqref when referring to equation numbers. When
\showonlyrefs is true, we monitor \eqref{(label)}, and those used will get an equa-
tion number, whereas those not used will not. Note: We do not monitor the normal
\ref command! See the follow paragraph for situations for what to do when \eqref
cannot be used to refer to the equation.

An equation where the tag is produced with a manual \tag* shouldn't be refer-
enced with the normal \eqref because that would format it according to the current
tag format. Using just \ref on the other hand may not be a good solution either as the
argument of \tag* is always set in upright shape in the equation and you may be ref-
encing it in italic text. In the example below, the command \refeq is used to avoid
what could possibly lead to confusion in cases where the tag font has very different
form in upright and italic shape (here we switch to Palatino in the example):

\begin{quote}
\renewcommand*{\rmdefault}{ppl}\normalfont\itshape
\begin{equation*}
a=b \label{eq:example}\tag*{Q\&A}
\end{equation*}
See \ref{eq:example} or is it better with \refeq{eq:example}?
\end{quote}

\begin{quote}
a = b
\text{Q\&A}
See Q\&A or is it better with Q\&A?
\end{quote}

Another problem sometimes faced is the need for showing the equation numbers
for only those equations actually referenced. In mathtools this can be done by set-
ting the key showonlyrefs to either true or false by using \mathtoolsset. You can also
choose whether or not to show the manual tags specified with \tag or \tag* by setting the option showmanualtags to true or false. For both keys just typing the name of it chooses true as shown in the following example.

\mathtoolsset{showonlyrefs,showmanualtags}
\usetagform{brackets}
\begin{gather}
a = a \label{eq:a} \\
b = b \label{eq:b} \tag{**}
\end{gather}
This should refer to the equation containing $a=a$: \eqref{eq:a}.

Then a switch of tag forms.
\usetagform{default}
\begin{align}
c &= c \label{eq:c} \\
d &= d \label{eq:d}
\end{align}
This should refer to the equation containing $d=d$: \eqref{eq:d}.

\begin{equation}
e = e
\end{equation}
Back to normal.\mathtoolsset{showonlyrefs=false}
\begin{equation}
f = f
\end{equation}

\begin{align*}
a &= a \quad [4] \\
b &= b \quad [**]
\end{align*}
This should refer to the equation containing $a=a$: [4]. Then a switch of tag forms.

\begin{align}
c &= c \\
d &= d
\end{align}
This should refer to the equation containing $d=d$: (5).

\begin{equation}
e = e
\end{equation}
Back to normal.

\begin{equation}
f = f
\end{equation}

Note that this feature only works if you use \eqref or \refeq to reference your equations.

When using showonlyrefs it might be useful to be able to actually add a few equation numbers without directly referring to them.

\begin{verbatim}
\noeqref{\textlangle\textrangle}
\end{verbatim}
The syntax is somewhat similar to \nocite. If a label in the list is undefined we will throw a warning in the same manner as \ref.

\footnote{I recommend setting showmanualtags to true, else the whole idea of using \tag doesn’t really make sense, does it?}
BUG 1: Unfortunately the use of the showonlyref introduces a bug within amsmath’s typesetting of formula versus equation number. This bug manifests itself by allowing formulas to be typeset close to or over the equation number. Currently no general fix is known, other than making sure that one’s formulas are not long enough to touch the equation number.

To make a long story short, amsmath typesets its math environments twice, one time for measuring and one time for the actual typesetting. In the measuring part, the width of the equation number is recorded such that the formula or the equation number can be moved (if necessary) in the typesetting part. When showonlyref is enabled, the width of the equation number depends on whether or not this number is referred to. To determine this, we need to know the current label. But the current label is not known in the measuring phase. Thus the measured width is always zero (because no label equals not referred to) and therefore the typesetting phase does not take the equation number into account.

BUG 2: There is a bug between showonlyrefs and the ntheorem package, when the ntheorem option thmmarks is active. The shown equation numbers may come out wrong (seems to be multiplied by 2). Or the end marker may be placed in the wrong place if a proof ends with a displayed formula, and that formula is not referred to.

There are two possible solutions to this both involving empheq. The easiest fix is to add the following line
\usepackage[overload,ntheorem]{empheq}
before loading ntheorem. But the overload option of course disables things like \intertext and \shortintertext.

The other thing to try is to drop the overload option and use empheq on the very last expression, as in
\begin{empheq}{align}
A & = B \label{eq:32} \\
& = 1. \label{eq:31}
\end{empheq}
\end{proof}

The empheq package fixes some problems with ntheorem and lets mathtools get correct access to the equation numbers again.

3.3 Extensible symbols
The number of horizontally extensible symbols in standard \LaTeX{} and amsmath is somewhat low. This part of the manual describes what mathtools does to help this situation.

3.3.1 Arrow-like symbols

\begin{verbatim}
\xleftrightarrow{\langle sup\rangle} \xRightarrow{\langle sup\rangle} \\
\xLeftarrow{\langle sup\rangle} \xLeftrightarrow{\langle sup\rangle} \\
\xhookleftarrow{\langle sup\rangle} \xhookrightarrow{\langle sup\rangle} \\
\xmapsto{\langle sup\rangle}
\end{verbatim}

Extensible arrows are part of amsmath in the form of the commands
\xrightarrow{\text{subscript}}{\text{superscript}} \text{ and } \xleftarrow{\text{subscript}}{\text{superscript}}

But what about extensible versions of say, \leftarrow or \Longleftarrow? It turns out that the above mentioned extensible arrows are the only two of their kind defined by amsmath, but luckily mathtools helps with that. The extensible arrow-like symbols in mathtools follow the same naming scheme as the one’s in amsmath so to get an extensible \leftarrow you simply do a

\[
A \xleftarrow{\text{under}}{\text{over}} B
\]

A over ⇐ = = = ⇒ below B

Caveat for \xmapsto: Whenever unicode-math is used, the macro \xmapsto will not work correctly (a random 7 appears). This is due to how we define \xmapsto: the symbol » (via \mapstochar, setup in the \LaTeX kernel) and extensible \rightarrow. The symbol behind \mapstochar is not defined in unicode, and thus is not available in the unicode math fonts.

\[\begin{array}{ll}
\xrightarpoondown{\text{sub}}{\text{sup}} & \xrightarpoonup{\text{sub}}{\text{sup}} \\
\xleftharpoondown{\text{sub}}{\text{sup}} & \xleftharpoonup{\text{sub}}{\text{sup}} \\
\xrightleftharpoons{\text{sub}}{\text{sup}} & \xleftrightharpoons{\text{sub}}{\text{sup}}
\end{array}\]

mathtools also provides the extensible harpoons shown above. They’re taken from [6]. For those liking visuals, here are the macros in use in the order listed in the two boxes above.

\[
\begin{array}{ll}
A \xrightarrow{\text{above}}{\text{below}} B & A \xrightarrow{\text{above}}{\text{below}} B \\
A \xrightarrow{\text{above}}{\text{below}} B & A \xrightarrow{\text{above}}{\text{below}} B \\
A \xrightarrow{\text{above}}{\text{below}} B & A \xrightarrow{\text{above}}{\text{below}} B \\
A \xrightarrow{\text{above}}{\text{below}} B & A \xrightarrow{\text{above}}{\text{below}} B \\
A \xrightarrow{\text{above}}{\text{below}} B & A \xrightarrow{\text{above}}{\text{below}} B \\
\end{array}\]

Change in 2020: In \xLeftarrow, \xRightarrow and \xLeftrightarrow we added a space to the argument(s), making the arrow slightly longer and moving the argument away from the large arrow head.

\footnote{The symbol \text{"--} is in unicode and unicode-math maps \xmapsto onto this symbol in the unicode math font. Whereas without unicode-math, \xmapsto is a symbol constructed by combining \mapstochar and \rightarrow. On the other hand luatex does have some special macros to create extensible arrows, but \textit{none} of these can write above and below at the same time. So for now we leave this unresolved.}
Similar to \right/\leftarrow but based on the long versions. For short arguments it stays the same length as the \long ... base arrow.

A \rightarrow B
\begin{align*}
&\frac{a}{b} B \\
&\frac{\text{above}}{\text{below}} B
\end{align*}

A \leftrightarrow B
\begin{align*}
&\frac{a}{b} B \\
&\frac{\text{above}}{\text{below}} B
\end{align*}

Similar to \Right/\Leftarrow but based on the long versions. For short arguments it stays the same length as the \long ... base arrow.

A \Rightarrow B
\begin{align*}
&\frac{a}{b} B \\
&\frac{\text{above}}{\text{below}} B
\end{align*}

A \Leftarrow B
\begin{align*}
&\frac{a}{b} B \\
&\frac{\text{above}}{\text{below}} B
\end{align*}

3.3.2 Braces and brackets

\LaTeX{} defines other kinds of extensible symbols like \overbrace and \underbrace, but sometimes you may want another symbol, say, a bracket.

\underbracket{rule thickness}\underbracket{bracket height}\underbracket{arg}
\overbracket{rule thickness}\overbracket{bracket height}\overbracket{arg}

The commands \underbracket and \overbracket are inspired by [6], although the implementation here is slightly different. Used without the optional arguments the bracket commands produce this:

$$\underbracket{foo\ bar}_{baz}$$

$$\overbracket{foo\ bar}^{baz}$$

The default rule thickness is equal to that of \underbrace (app. 5/18 ex) while the default bracket height is equal to app. 0.7 ex. These values give really pleasing results in all font sizes, but feel free to use the optional arguments. That way you may get “beauties” like

\begin{verbatim}
  \underbracket[3pt]{xxx\ yyy}_{zzz} \quad \overbracket[1pt][7pt]{xxx\ yyy}_{zzz}
\end{verbatim}
The standard implementation of the math operators \underbrace and \overbrace in \LaTeX has some deficiencies. For example, all lengths used internally are fixed and optimized for 10 pt typesetting. As a direct consequence thereof, using font sizes other than 10 will produce less than optimal results. Another unfortunate feature is the size of the braces. In the example below, notice how the math operator \sum places its limit compared to \underbrace.

\[ \underbrace{\langle \text{arg} \rangle} \quad \LaTeX \underbrace{\langle \text{arg} \rangle} \]

\[ \overbrace{\langle \text{arg} \rangle} \quad \LaTeX \overbrace{\langle \text{arg} \rangle} \]

The blue lines indicate the dimensions of the math operator and the green lines the dimensions of \footnotesize. As you can see, there seems to be too much space between the brace and the \texttt{zzz} whereas the space between brace and \texttt{foof} is okay. Let’s see what happens when we use a bigger font size:

\[ \sum_n \texttt{foof} \]

Now there’s too little space between the brace and the \texttt{zzz} and also too little space between the brace and the \texttt{foof}. If you use Computer Modern you’ll actually see that the \texttt{f} overlaps with the brace! Let’s try in \footnotesize:

\[ \sum_n \texttt{foof} \]

Here the spacing above and below the brace is quite excessive.

As \overbrace has the exact same problems, there are good reasons for mathtools to make redefinitions of \underbrace and \overbrace. These new versions work equally well in all font sizes and fix the spacing issues and apart from working with the default Computer Modern fonts, they also work with the packages mathpazo, pamath, fourier, euler, cmbrght, and mathptmx. If you use the cfonts to get the full Concrete fonts, the original version saved under the names \LaTeX\underbrace and \LaTeX\overbrace is better, due to the special design of the Concrete extensible braces. In that case you should probably just add the lines

\[
\let\underbrace\LaTeXunderbrace
\let\overbrace\LaTeXoverbrace
\]

to your preamble after loading mathtools which will restore the original definitions of \overbrace and \underbrace.

### 3.4 New mathematical building blocks

In this part of the manual, various mathematical environments are described.
3.4.1 Matrices

All of the amsmath matrix environments center the columns by default, which is not always what you want. Thus mathtools provides a starred version for each of the original environments. These starred environments take an optional argument specifying the alignment of the columns, so that

\[
\begin{pmatrix*}[r] 
-1 & 3 \\
2 & -4 
\end{pmatrix*}
\]

yields

\[
\begin{pmatrix}
-1 & 3 \\
2 & -4 
\end{pmatrix}
\]

The optional argument (default is [c]) can be any column type valid in the usual array environment.

While we are at it, we also provide fenced versions of the smallmatrix environment. To keep up with the naming of the large matrix environments, we provide both a starred and a non-starred version. Since smallmatrix is defined in a different manner than the matrix environment, the option to say smallmatrix* has to be either c, l or r. The default is c, which can be changed globally using the smallmatrix-align = (c,l or r).

\[
\begin{bsmallmatrix*}[r] 
a & -b \\
-c & d 
\end{bsmallmatrix*}
\]

```latex
\begin{bsmallmatrix}
a & -b \\
-c & d 
\end{bsmallmatrix}
```
yields
\[
\begin{bmatrix}
a & -b \\
-c & d
\end{bmatrix}
\]

Inside the \texttt{Xsmallmatrix} construction a small space is inserted between the fences and the contents, the size of it can be changed using \texttt{smallmatrix-inner-space}=(\textit{some spacing command}), the default is \texttt{,}.

As an extra trick the fences will behave as open and closing fences despite their auto-scaling nature.\footnote{The helper macro \texttt{\Framebox{\ldots}} is equivalent to \texttt{\framebox{\footnotesize\strut\ldots}}}

\subsection{The multlined environment}

\begin{Verbatim}
\begin{multlined}[(pos)][(width)](contents)\end{multlined}
\shoveleft[(dimen)]{(arg)} \shoveright[(dimen)]{(arg)}
\texttt{firstline-afterskip = (dimen)} \texttt{lastline-preskip = (dimen)}
\texttt{multlined-width = (dimen)} \texttt{multlined-pos = c|b|t}
\end{Verbatim}

Some of the amsmath environments exist in two forms: an outer and an inner environment. One example is the pair \texttt{gather} & \texttt{gathered}. There is one important omission on this list however, as there is no inner \texttt{multlined} environment, so this is where \texttt{mathtools} steps in.

One might wonder what the sensible behavior should be. We want it to be an inner environment so that it is not wider than necessary, but on the other hand we would like to be able to control the width. The current implementation of \texttt{multlined} handles both cases. The idea is this: Set the first line flush left and add a hard space after it; this space is governed by the \texttt{firstline-afterskip} key. The last line should be set flush right and preceded by a hard space of size \texttt{lastline-preskip}. Both these hard spaces have a default value of \texttt{\multlinegap}. Here we use a \texttt{t} in the first optional argument denoting a top-aligned building block (the default is \texttt{c}).\footnote{\texttt{\left} and \texttt{\right} creates an \textit{inner} construction, and not as one might expect something where a preceding \texttt{\sin} sees an opening fence, thus the space before or after may be too large. Inside this construction they behave.}

\[
A = \begin{multlined}[t]
\Framebox[4cm]{first} \\ \\
\Framebox[4cm]{last}
\end{multlined} B
\]

Note also that \texttt{multlined} gives you access to an extended syntax for \texttt{\shoveleft} and \texttt{\shoveright} as shown in the example below.

\[
\begin{multlined}
\Framebox[.6\columnwidth]{First line} \\ \\
\Framebox[.5\columnwidth]{Second line} \\ \\
\shoveleft{L+E+F+T}
\end{multlined}
\]

\begin{center}
\begin{tabular}{|c|}
\hline
first \\
\hline
last \\
\hline
\end{tabular}
\end{center}

\begin{tabular}{|c|}
\hline
A = \\
\hline
\begin{tabular}{|c|}
\hline
first \\
\hline
\end{tabular} \\
\hline
\end{tabular}

\begin{tabular}{|c|}
\hline
B \\
\hline
\end{tabular}
You can also choose the width yourself by specifying it as an optional argument:

\[
\begin{multlined}[b][7cm]
\text{first} \\
\text{last} = B
\end{multlined}
\]

There can be two optional arguments (position and width) and they’re interchangeable.

**Bug 1:** If used inside an array or a derivative (say, a matrix variant), multlined does not work as expected. The implementation contains an ‘invisible’ line after the first multiline row, inside an array this line is no longer ‘invisible’ because array sets \baselineskip to zero. We have no general workaround for this.

**Bug 2:** Due to the way multlined is implemented, certain constructions do not work inside multlined. We have added a hook (\MultlinedHook) that can be added to. The default value is a fix for subarray and crampedsubarray and thus for \substack and \crampedsubstack (and a few others, thus add to the hook, don’t replace it).

This can actually be used to fix Bug 1:

\usepackage{mathtools,etoolbox}
\newlength\Normalbaselineskip
\setlength\Normalbaselineskip{\baselineskip}
\appto\MultlinedHook{
    \setlength\baselineskip{\Normalbaselineskip}
}

**Caveat:** \shoveleft and \shoveright does not work as expected if used as the last line in multlined.
### 3.4.3 More cases-like environments

Anyone who has tried to use an integral in the regular cases environment from ams-math will have noticed that it is set as

\[
a = \begin{cases}
E = mc^2 & \text{Nothing to see here} \\
\int x-3\, dx & \text{Integral is text style}
\end{cases}
\]

mathtools provides two environments similar to cases. Using the dcases environment you get the same output as with cases except that the rows are set in display style.

\[
\begin{dcases}
E = mc^2 & c \approx 3.00 \times 10^8 \text{m/s} \\
\int x-3\, dx & \text{Integral is display style}
\end{dcases}
\]

Additionally the environment dcases* acts just the same, but the second column is set in the normal roman font of the document.\(^{10}\)

\[
a = \begin{dcases*}
E = mc^2 & \text{Nothing to see here} \\
\int x-3\, dx & \text{Integral is display style}
\end{dcases*}
\]

The environments rcases, rcases*, drcases and drcases* are equivalent to cases and dcases, but here the brace is placed on the right instead of on the left.

\[
\begin{rcases}
x^2 & \text{for } x>0 \\
x^3 & \text{else}
\end{rcases} \quad \Rightarrow \cdots
\]

\(^{10}\)Or rather: it inherits the font characteristics active just before the dcases* environment.
3.4.4 Emulating indented lines in alignments

\MoveEqLeft[(number)]

In [7], Ellen Swanson recommends that whenever one has a long displayed formula, spanning several lines, and it is unfeasible to align against a relation within the first line, then all lines in the display should be aligned at the left most edge of the first line, and all subsequent lines should be indented by 2 em (or if needed by a smaller amount). That is we are talking about displays that end up looking like this:\n
\begin{align*}
\MoveEqLeft \text{Long first line} \\
&= \text{2nd line} \\
&\leq \ldots
\end{align*}

Traditionally one could do this by starting subsequent lines by & \quad \ldots, but that is tedious. Instead the example above was made using \MoveEqLeft:

\begin{align*}
\MoveEqLeft \text{Long first line} \\
= \text{2nd line} \\
\leq \ldots
\end{align*}

\MoveEqLeft is placed instead of the & on the first line, and will effectively move the entire first line [(number)] of ems to the left (default is 2). If you choose to align to the right of the relation, use \MoveEqLeft[3] to accommodate the extra distance to the alignment point:

\begin{align*}
\MoveEqLeft[3] \text{Part 1} \\
= {} & \text{2nd line} \\
+ & \text{last part}
\end{align*}

\MoveEqLeft[3] is placed instead of the & on the first line, and will effectively move the entire first line [(number)] of ems to the left (default is 2). If you choose to align to the right of the relation, use \MoveEqLeft[3] to accommodate the extra distance to the alignment point:

\begin{align*}
\MoveEqLeft[3] \text{Part 1} \\
= {} & \text{2nd line} \\
+ & \text{last part}
\end{align*}

\MoveEqLeft[]: If the first part of the equation starts with say [a], \MoveEqLeft may attempt to eat it! You can prevent this by specifying the optional argument (remember the default is the same as \MoveEqLeft[2]) or by using \MoveEqLeft{}.

Caveat regarding \MoveEqLeft: If the first part of the equation starts with say [a], \MoveEqLeft may attempt to eat it! You can prevent this by specifying the optional argument (remember the default is the same as \MoveEqLeft[2]) or by using \MoveEqLeft{}.

3.4.5 Boxing a single line in an alignment

The amsmath package provides the \boxed macro to box material in math mode. But this of course will not work if the box should cross an alignment point. We provide a macro that can.
\begin{align*}
\Aboxed{\text{left hand side} & \text{ right hand side}}

\text{Example}
\begin{align*}
\Aboxed{f(x) & = \int h(x) \, dx} & \Aboxed{g(x)}
\end{align*}
\end{align*}

One can have multiple boxes on each line, and the “& right hand side” can even be missing. Here is an example of how the padding in the box can be changed.
\begin{align*}
\setlength{\fboxsep}{1em}
\Aboxed{f(x) &= 0} & \Aboxed{g(x) &= b} \\
\Aboxed{h(x)} & \Aboxed{i(x)}
\end{align*}

Note how the \fboxsep change only affects the box coming immediately after it.

As \Aboxed looks for the alignment & it may be necessary to hide constructions like matrices that also make use of &. Just add a set of braces around the construction you want to hide.

\textbf{Tip:} At times users want a \Aboxed with a different type of box. To help we provide a \Aboxed builder:

\MakeAboxedCommand\Aboxedname\boxmacro

where \boxname should be a macro taking a single argument. The normal \Aboxed is the same as \MakeAboxedCommand\Aboxed\fbox. Thus you can do
\begin{align*}
\Aboxed{f(x) = x^2} & \Aboxed{f(x) = x^2}
\end{align*}

\newcommand{\myboxA}[1]{\fcolorbox{gray}{yellow}{#1}}

\footnote{Maintainer (daleif)’s note: This is one of my most used constructions whenever I edit manuscripts. I’d rather use this than for example \multline as I find it gives the text a more consistent look.}

\footnote{Note that internally \Aboxed does use \boxed.}
3.4.6 Adding arrows between lines in an alignment

This first macro is a bit misleading, it is only intended to be used in combination with the alignat(*) environment.

To add, say ⇕ between two lines in an alignment use \ArrowBetweenLines and the alignat environment (note the extra pair of &’s in front):

\begin{alignat}{2}
&\text{\ArrowBetweenLines} & \text{\framebox[1.5cm]{}} &= \text{\framebox[3cm]{}} \\
&& \text{\ArrowBetweenLines*} & \text{\framebox[1.5cm]{}} &= \text{\framebox[2cm]{}}
\end{alignat}

resulting in

\begin{tabular}{c}
\hline
\end{tabular}

(7) \hspace{2cm} \hspace{2cm}
\begin{tabular}{c}
\hline
\end{tabular}

(8)

Note the use of && starting each regular line of math. For adding the arrow on the right, use \ArrowBetweenLines*[\textit{symbol}], and end each line of math with &&.

\begin{alignat*}{2}
\text{\ArrowBetweenLines*} & \text{\downarrow} & \text{\framebox[1.5cm]{}} &= \text{\framebox[2cm]{}} \\
&& & \text{\ArrowBetweenLines*[\textit{Downarrow}]} & \text{\framebox[1.5cm]{}} &= \text{\framebox[2cm]{}}
\end{alignat*}

resulting in

\begin{tabular}{c}
\hline
\end{tabular}


Feature request by Bruno Le Floch (and many others) 2011/01/25

3.4.7 Centered \vdots

If one wants to mark a vertical continuation, there is the \vdots command, but combine this with an alignment and we get something rather suboptimal

\begin{tabular}{c}
\hline
\end{tabular}

\vdots

\begin{tabular}{c}
\hline
\end{tabular}
It would be nice to have (1) a \( \vdots \) centered within the width of another symbol, and (2) a construction similar to \( \ArrowBetweenLines \) that does not take up so much space. We provide both.

\[
\begin{align*}
a &= b \\
& \vdotswithin{=} \\
& = c \\
& \shortvdotswithin{=} \\
& = d
\end{align*}
\]

yielding

\[
a = b \\
\vdots \\
= c \\
\vdots \\
= d
\]

Thus \( \vdotswithin{=} \) creates a box corresponding to \( {}={} \) and typesets a “…” centered inside it. When doing this as a normal line in an alignment leaves us with excessive space which \( \shortvdotswithin{=} \) takes care with for us.

The macro call \( \shortvdotswithin{=} \) corresponds to

\[
\MTFlushSpaceAbove \\
& \vdotswithin{=} \\
\MTFlushSpaceBelow
\]

whereas \( \shortvdotswithin{*}{=} \) is the case with

\[
\MTFlushSpaceAbove \\
& \vdotswithin{=} & \\
\MTFlushSpaceBelow
\]

Note how \( \MTFlushSpaceBelow \) implicitly adds a \( \backslash \) at the end of the line. Thus one cannot have more on the line when using \( \shortvdotswithin{=} \) or the starred version. But, if needed, one can de-construct the macro and arrive at

\[
\begin{alignat*}{3}
A &+ B & &\&= C & &\&+ D \\
\MTFlushSpaceAbove \\
&\vdotswithin{+} &\& &\vdotswithin{+}
\end{alignat*}
\]
\MTFlushSpaceBelow
C + D &
= Y &
+ K
\end{alignat*}

yielding

\begin{align}
A + B &= C + D \\
C + D &= Y + K
\end{align}

If one has the need for such a construction.

The de-spaced version does support the \texttt{spreadlines} environment. The actual amount of space being \textit{flushed} above and below can be controlled by the user using the two options indicated. Their original values are 2.15\texttt{origjot} and \texttt{origjot} respectively (\texttt{origjot} is usually 3pt).

### 3.5 Intertext and short intertext

\begin{align}
a &= b \\
& \shortintertext{Some text}
c &= d
\end{align}

(a = b) (9)

Some text

\begin{align}
c &= d
\end{align}

(c = d) (10)

Using the command \texttt{\shortintertext} alleviates this situation somewhat:

\begin{align}
a &= b \\
& \shortintertext{Some text}
c &= d
\end{align}

(a = b) (11)

Some text

\begin{align}
c &= d
\end{align}

(c = d) (12)

It turns out that both \texttt{\shortintertext} and the original \texttt{\intertext} from \texttt{amsmath} has a slight problem. If we use the \texttt{spreadlines} (see section 4.4) to open up the equations in a multiline calculation, then this opening up value also applies to the spacing above and below the original \texttt{\shortintertext} and \texttt{\intertext}. It can be illustrated
using the following example, an interested reader can apply it with and without the original \intertext and \shortintertext.

% the original \intertext and \shortintertext
\mathtoolsset{original-intertext,original-shortintertext}
\newcommand\myline{\par
\noindent\rule{\textwidth}{1mm}}
\myline
\begin{spreadlines}{1em}
\begin{align*}
AA\\ BB\\ \intertext{\myline}
AA\\ BB\\ \shortintertext{\myline}
AA\\ BB
\end{align*}
\end{spreadlines}
\myline
We now fix this internally for both \intertext and \shortintertext, plus we add the possibility to fine-tune spacing around these constructions. The original versions can be brought back using the original-x keys below.

\begin{itemize}
\item \intertext{(text)}
\item \shortintertext{(text)}
\end{itemize}
\begin{itemize}
\item original-intertext = true|false (default: false)
\item original-shortintertext = true|false (default: false)
\item above-intertext-sep = (dimen) (default: 0pt)
\item below-intertext-sep = (dimen) (default: 0pt)
\item above-shortintertext-sep = (dimen) (default: 3pt)
\item below-shortintertext-sep = (dimen) (default: 3pt)
\end{itemize}

The updated \shortintertext will look like the original version unless for areas with an enlarged \jot value (see for example the spreadlines, section 4.4). Whereas \intertext will have a slightly smaller value above and below (corresponding to about 3pt less space above and below), the spacing around \intertext should now match the normal spacing going into and out of an align.

***Tip***: \intertext and \shortintertext also works within gather.

### 3.6 Paired delimiters

Feature request by Lars Madsen
2004/06/25

In the amsmath documentation it is shown how to define a few commands for typesetting the absolute value and norm. These definitions are:

\begin{verbatim}
\newcommand*{\abs}[1]{\lvert#1\rvert}
\newcommand*{\norm}[1]{\lVert#1\rVert}
\end{verbatim}

While they produce correct horizontal spacing you have to be careful about the vertical spacing if the argument is just a little taller than usual as in

\[ \frac{a}{b} \]

Here it won’t give a nice result, so you have to manually put in either a \texttt{\left–\right} pair or a \texttt{\bigl–\bigr} pair. Both methods mean that you have to delete your \abs command, which may not sound like an ideal solution.
With the command \DeclarePairedDelimiter you can combine all these features in one easy to use command. Let’s show an example:

\DeclarePairedDelimiter\abs{\lvert}{\rvert}

This defines the command \abs just like in the amsmath documentation but with a few additions:

- A starred variant: \abs* produces delimiters that are preceded by \left and \right resp.:

  \[
  \abs*{\frac{a}{b}}
  \]
  \[
  \frac{a}{b}
  \]

- A variant with an optional argument: \abs[\text{size cmd}]\{\text{body}\}, where \text{size cmd} is either \text{big}, \text{Big}, \text{bigg}, or \text{Bigg} (if you have any biggger versions you can use them too).

  \[
  \abs[\text{Bigg}]\{\frac{a}{b}\}
  \]
  \[
  \frac{a}{b}
  \]

**A note on the delimiters.** Whenever the manual scalings are used we internally convert them into their \ldots l and \ldots r variants. Therefore if you define your own manual scalers, remember to also have -l and -r variants.

*New from version 1.30. You can now use » « (dot) as an empty delimiter, for macros what only have a delimiter on one side.*

\[
\DeclarePairedDelimiterX{\abs[\text{cmd}]\{\text{num args}\}\{\text{left delim}\}\{\text{right delim}\}\{\text{body}\}}{\text{delimsize}}
\]

Sometimes one might want to have the capabilities of \DeclarePairedDelimiter, but also want a macro which takes more than one argument plus being able to specify the body of the generated macro.

\DeclarePairedDelimiterX extends the features of \DeclarePairedDelimiter such that the user will get a macro which is fenced off at either end, plus the capability to provide the \text{body} for whatever the macro should do within these fences.

Inside the \text{body} part, the macro \text{delimsize} refers to the size of the outer fences. It can then be used inside \text{body} to scale any inner fences.

Thus

\[
\DeclarePairedDelimiter\abs[\text{cmd}]\{\text{left delim}\}\{\text{right delim}\}
\]

is the same thing as

\[13\text{The syntax is borrowed from }\left.\frac{1}{2}\right]\rvert\]

26
\DeclarePairedDelimiterX{\innerp}[2]{\langle}{\rangle}{#1}

Let us do some examples. First we want to prepare a macro for inner products, with two arguments such that we can hide the character separating the arguments (a journal style might require a semi-colon, so we will save a lot of hand editing). This can be done via

\DeclarePairedDelimiterX{\innerp}[2]{\langle}{\rangle}{#1,#2}

More interestingly we can refer to the size inside the \langle code\rangle. Here we do a weird three argument 'braket'

\DeclarePairedDelimiterX{\braket}[3]{\langle}{\rangle}{#1\mathopen{}#2\mathopen{}#3}

Note the use of \mathopen{} in the \langle body\rangle of \braket, this is very important when a scalable delimiter is being used and it does not present itself as a left or right delimiter. You will see why it is needed if you use \braket{A}{-B}{C} in a version without \mathopen{}.

Then we get

\[
\innerp{A}{\frac{1}{2}} \quad \braket{B}{\sum_{k} f_k}{C}
\]

\[
\left\langle A, \frac{1}{2} \right\rangle \left\langle B \middle| \sum_{k} f_k \right\rangle C
\]

With the inner scaling, we can provide macros whose syntax closely follows the mathematical meaning. For building sets, try this

% just to make sure it exists
\providecommand{\given}{}
% can be useful to refer to this outside \Set
\newcommand{\SetSymbol}[1][\delimsize]{% 
\nonscript:\#1\vert 
\allowbreak 
\nonscript:: 
\mathopen{}}
\DeclarePairedDelimiterX{\Set}[1][\{}{\}}{\SetSymbol\allowbreak \given(\SetSymbol[\delimsize])}

% The reason for using a separate \SetSymbol macro has to do with complicated set definitions, where the condition spans several lines. In this case \Set cannot be used. Thus it is nice to be able to refer to the specific set building symbol we have decided to use in this document. Also remember to add \allowbreak before the proceeding inserted space. Then that space will disappear when a line break occurs.

\[
\Set*{ x \in X \given \frac{\sqrt{x}}{x^2+1} > 1 } \]

\footnote{\textsuperscript{14}Basically, the problem is that \vert is a 'symbol', thus \[ - B \] is interpreted subtraction, not a symbol followed by negative \( B \) (aka a sign). When \mathopen{} is added, \( B \) is told that an opening delimiter just occurred, and it will adjust the minus accordingly.}

\footnote{\textsuperscript{15}The reason for using a separate \SetSymbol macro has to do with complicated set definitions, where the condition spans several lines. In this case \Set cannot be used. Thus it is nice to be able to refer to the specific set building symbol we have decided to use in this document. Also remember to add \allowbreak before the proceeding inserted space. Then that space will disappear when a line break occurs.}
\[
\left\{ x \in X \bigg| \frac{\sqrt{x}}{x^2 + 1} > 1 \right\}
\]

Thus we end up with a syntax much closer to how we read this aloud. Also we hide the ‘given’ symbol for easy replacement.\(^\text{16}\)

Combining with etoolbox it becomes easy to make a function that automatically provides a marker for a blank argument:

\begin{verbatim}
\usepackage{etoolbox}
\DeclarePairedDelimiterX{\norm}[1]\Vert\Vert{\ifblank{#1}{\cdot}{#1}}
\end{verbatim}

Then \(\norm{}\) will give you \(\|\cdot\|\) while \(\norm{a}\) gives the expected \(\|a\|\).

\begin{verbatim}
\DeclarePairedDelimiterXPP{\〈}{\cmd}{\num args}{\pre code}{\left_delim}{\right_delim}{\post code}{\body}
\end{verbatim}

\(\DeclarePairedDelimiterX\lnorm{\lVert}{\rVert}{\mathchoice{\big}{\big}{\mathopen{\bigm|}}\mathopen{\bigm|}}\)

An \(L^2\) norm can now be defined as

\begin{verbatim}
\DeclarePairedDelimiterXPP{\Prob}{\mathbb{P}}{\given}{\bigl}{\rVert}{\mathopen{\bigm|}}\)
\end{verbatim}

A probability macro with built-in support for conditionals (\texttt{\given} initialized as above)

\begin{verbatim}
\DeclarePairedDelimiterXPP{\Prob}{\mathbb{P}}{\given}{\mathchoice{\mathopen{\bigm|}}\mathopen{\bigm|}}\)
\end{verbatim}

Thus giving support for \(\Prob{A \given B}\).

\textbf{Note 1:} As the number of arguments increases the \(\DeclarePairedDelimiter\)... macros become hard for users to understand. A key-value interface would be better. This is planned for a future release. In \url{http://tex.stackexchange.com/a/3929} there is a suggested replacement for \(\ DeclarePairedDelimiter\), that greatly reduces the number of macros and provides a key-val interface. However, the code uses xparse, and if we want to use xparse for some of our macros, we might just as well rewrite the entire mathtools package in expl3. Also it is not obvious how to get xparse to support \([\text{3}]\) for the number of arguments. We will consider this for a future release.

\textbf{Note 2:} If you want to define your own manual scaler macros, it is important that you besides \textbackslash foo also define \textbackslash fool and \textbackslash foor. When a scaler is specified, in say \textbackslash abs[\big\l]{\arg}, we actually use \textbackslash big\l and \textbackslash bigr.

\(^{16}\) The \texttt{\nonscript} construction removes the \: in sub- and superscript, this might not always be preferable. You can use \texttt{\mathchoice\:} instead of \texttt{\nonscript\:}.

\(^{17}\) The added ‘\_2’ is hard to get around the argument parsing.

\(^{18}\) Extended \texttt{\DeclarePairedDelimiter} with Pre and Post code.
Note 3: In the unlikely event that you choose to use, say, \section{Foo $\norm\big\{bar\}$} you will get an error because \big is getting expanded too early. Sadly, due to how \DeclarePairedDelimiter is implemented one cannot use [\protect\big]. At present (until amsmath makes \big and friends robust), the best solution\textsuperscript{19} is to load the etoolbox package and add \robustify\big and similar to your preamble.

3.6.1 Expert use

Within the starred version of \DeclarePairedDelimiter and \DeclarePairedDelimiterX we make a few changes such that the auto scaled \left and \right fences behave as opening and closing fences, i.e. \sin(x) vs. \sin(x) (the later made via $\sin\left(x\right)$), notice the gap between `sin' and `. In some special cases it may be useful to be able to tinker with the behavior.

Internally several macros are created, including three call backs that take care of the formatting. There is one internal macro for the starred version, labeled \star, the other two are labeled nostarnonscaled and nostarscaled. Within \langle\langle\text{code}\rangle\rangle, \#1 will be replaced by the (scaled) left fence, \#3 the corresponding (scaled) right fence, and \#2 the stuff in between. For example, here is how one might turn the content into \mathinner:

\reDeclarePairedDelimiterInnerWrapper{\langle\langle\langle\text{code}\rangle\rangle\rangle}{\langle\langle\text{macro name}\rangle\rangle}{\langle\langle\star\rangle\rangle}{\langle\langle\{\text{star or nostarnonscaled or nostarscaled}\}\rangle\rangle}{\langle\langle\{\text{code}\}\rangle\rangle}

The default values for the call backs correspond to

\star: \mathopen{\langle\langle\langle\text{code}\rangle\rangle\rangle} \mathclose{\langle\langle\langle\text{code}\rangle\rangle\rangle}\aftergroup\egroup

nostarnonscaled: \ifx.#1\else\mathopen{\langle\langle\langle\text{code}\rangle\rangle\rangle}\fi\#2\ifx.#3\else\mathclose{\langle\langle\langle\text{code}\rangle\rangle\rangle}\fi

nostarscaled: \mathopen{\langle\langle\langle\text{code}\rangle\rangle\rangle}\#1\#2\mathclose{\langle\langle\langle\text{code}\rangle\rangle\rangle}\#3

The two nostar... versions look the same, but they are not. In most (math) fonts, the first item in this list will be different from the rest (the superscript sits higher).\textsuperscript{20}

\mathclose{\langle\langle\langle\text{code}\rangle\rangle\rangle}\#2\mathclose{\langle\langle\langle\text{code}\rangle\rangle\rangle}\#3

Breaking change: Prior to May 2017, we used two wrappers, the other named nostar. As of May 2017 nostar has been split into nostarnonscaled and nostarscaled and nostar alone is no longer supported (will give an error).

Note: Since we are using macros to add the \left...\right constructions around some body, it is (in principle) possible to make such a construction breakable across lines, even breakable within an align construction. Currently, this can only be applied to macros made using \DeclarePairedDelimiter and not macros made using \DeclarePairedDelimiterX or \DeclarePairedDelimiterXPP as the contents are

\textsuperscript{19}https://tex.stackexchange.com/a/639238/3929

\textsuperscript{20}Interestingly it did not show up in the font of this manual, which uses the fourier font set.
typeset inside a group (to limit \delimsize) and thus hide any & or \ from align and align breaks down.

When \DeclarePairedDelimiter is used, Sebastien Gouezel has provided the following example

\newcommand\MTkillspecial[1]{% helper macro
  \bgroup
  \catcode`\&=9
  \let\relax
  \scantokens{#1}%
  \egroup
}
\DeclarePairedDelimiter\abs\lvert\rvert
\reDeclarePairedDelimiterInnerWrapper\abs{star}{
  \mathopen{#1\vphantom{\MTkillspecial{#2}}\kern-\nulldelimiterspace\right.}
  #2
  \mathclose{\left.\kern-\nulldelimiterspace\vphantom{\MTkillspecial{#2}}#3}}

Then this example works just fine:

\begin{align*}
  f(a) &= \begin{aligned}
    \abs*{ & \frac{1}{2}+\cdots \\
    & \ldots+\frac{1}{2} }
  \end{aligned}
\end{align*}

\begin{align*}
  f(a) &= \begin{aligned}
    \abs*{ & \frac{1}{2}+\cdots \\
    & \ldots+\frac{1}{2} }
  \end{aligned}
\end{align*}

3.7 Special symbols

This part of the manual is about special symbols. So far only one technique is covered, but more will come.

3.7.1 Left and right parentheses

\lparen \rparen

When you want a big parenthesis or bracket in a math display you usually just type

\left( ... \right) or \left[ ... \right]

\TeX also defines the macro names \lbrack and \rbrack to be shorthands for the left and right square bracket resp., but doesn’t provide similar definitions for the parentheses. Some packages need command names to work with\footnote{The \emph{empheq} package needs command names for delimiters in order to make auto-scaling versions.} so \mathtools defines the commands \lparen and \rparen to represent the left and right parenthesis resp.
3.7.2 Vertically centered colon

When trying to show assignment operations as in \( a := b \), one quickly notices that the colon is not centered on the math axis as the equal sign, leading to an odd-looking output. The command \( \texttt{\textbackslash vcentcolon} \) is a shorthand for such a vertically centered colon, and can be used as in $a \vcentcolon= b$ and results in the desired output: \( a := b \). See also the colonequals package.

Typing \( \texttt{\textbackslash vcentcolon} \) every time is quite tedious, so one can use the key \texttt{centercolon} to make the colon active instead.

\begin{verbatim}
\mathtoolsset{centercolon}
\[
\texttt{a := b}
\]
\mathtoolsset{centercolon=false}
\[a := b\]
\end{verbatim}

In this case the command \( \texttt{\textbackslash ordinarycolon} \) typesets an … ordinary colon (what a surprise).

**Warning:** \texttt{centercolon} does not work with languages that make use of an active colon, most notably \textit{French}. Sadly the \texttt{babel} package does not distinguish between text and math when it comes to active characters. Nor does it provide any hooks to deal with math. So currently no general solution exists for this problem.

3.7.3 Some extra symbols involving vertically centered colon

Inspired by the \texttt{txfonts} and \texttt{pxfonts} we provide some colon+symbol combinations such that they will be available even when current font does not provide the combined symbol.

**Breaking change Summer 2022:** As is listed in the manual for \texttt{newtxmath/newpxmath}\textsuperscript{22}, the symbols \( \texttt{\textbackslash coloneq} \backslash \text{\textbackslash Coloneq} \) \texttt{\textbackslash eqcolon} \backslash \text{\textbackslash Eqcolon} \) \texttt{\textbackslash colonapprox} \backslash \text{\textbackslash Colonapprox} \) \texttt{\textbackslash colonsim} \backslash \text{\textbackslash Colonsim} \) \texttt{\textbackslash colon} \backslash \text{\textbackslash Colon} \) \texttt{\textbackslash colondash} \backslash \text{\textbackslash Colondash} \) \texttt{\textbackslash approxcolon} \backslash \text{\textbackslash Approxcolon} \) \texttt{\textbackslash simcolon} \backslash \text{\textbackslash Simcolon} \) \texttt{\textbackslash dashcolon} \backslash \text{\textbackslash Dashcolon} \) \texttt{\textbackslash approxcolon} \backslash \text{\textbackslash Approxcolon} \) \texttt{\textbackslash simcolon} \backslash \text{\textbackslash Simcolon} \) \texttt{\textbackslash dashcolon} \backslash \text{\textbackslash Dashcolon} \)

\textsuperscript{22}These superceeds \texttt{txfonts} and \texttt{pxfonts}.
If for some reason you need to revert to original (pre summer 2022) colon symbol names, use the `legacycolonsymbols` package option. In which case the macro names maps to:

\[
\begin{align*}
\coloneqq & := \Coloneqq := \colononeq & \simcolon & := \simcolon \simcolon \simcolon := \\
\eqqcolon & := \Eqqcolon := \eqcolon \eqcolon \eqcolon := \\
\dblcolon & := \dblcolon \dblcolon \dblcolon \dblcolon :=
\end{align*}
\]

Extra symbols added Summer 2022:

\[
\begin{align*}
\approxcolon & \approxcolon \approxcolon \approxcolon := \\
\colondash & \colondash \colondash \colondash :=
\end{align*}
\]

Colon symbol names with `legacycolonsymbols` package option.

Note that the symbol names are added at the start of the document via the \provide... mechanism, thus if some font package provide a symbol of the same macro name, we do nothing.

### 3.7.4 A few additional symbols

Most provided math font sets are missing the symbols \( \nuparrow \) and \( \ndownarrow \) (i.e. negated up- and downarrow) plus a ‘big’ version of \( \times \). Therefore we will provide constructed versions of these whenever they are not already available.

\[
\begin{align*}
\nuparrow & \colon \nuparrow \nuparrow \nuparrow \nuparrow := \\
\ndownarrow & \colon \ndownarrow \ndownarrow \ndownarrow := \\
\bigtimes & \colon \bigtimes \bigtimes \bigtimes :=
\end{align*}
\]

Note: These symbols are constructed via features from the `graphicx` package, and thus may not display correctly in most DVI previewers. Also note that \( \nuparrow \) and \( \ndownarrow \) are constructed via \( \nrightarrow \) and \( \nleftarrow \) respectively, so these need to be present. Usually this is done via `amssymb`, but some packages may be incompatible with `amssymb` so the user will have to load `amssymb` or a similar package, that provides \( \nrightarrow \) and \( \nleftarrow \), themselves.

With those requirements in place, we have

\[
\begin{align*}
\lim_{a \downarrow 0} f(a) & \neq \bigtimes_n X_n \\
\frac{ \bigtimes_{k=1}^7 B_k \nuparrow \Omega }{2} & \quad \lim_{a \downarrow 0} f(a) \neq \bigtimes_n X_n \\
\frac{ \bigtimes_{k=1}^7 B_k \nuparrow \Omega }{2} & \quad \lim_{a \downarrow 0} f(a) \neq \bigtimes_n X_n
\end{align*}
\]

### 4 A tribute to Michael J. Downes

Michael J. Downes (1958–2003) was one of the major architects behind `amsmath` and member of the `\LaTeX Team. He made many great contributions to the `\LaTeX` community; not only by the means of widely spread macro packages such as `amsmath` but also in the form of actively giving advice on newsgroups. Some of Michael’s macro solutions on the newsgroups never made it into publicly available macro packages although they certainly deserved it, so `mathtools` tries to rectify this matter. The macros described in this section are either straight copies or heavily inspired by his original postings.
4.1 Mathematics within italic text

\texttt{mathic=true|false}

\LaTeX{} usually takes care of italic corrections in text, but fails when it comes to math. If you use the \LaTeX{} inline math commands $\textbackslash ( \text{and} \textbackslash )$ you can however work around it by setting the key \texttt{mathic} to true as shown below.

\begin{quote}
\itshape
Compare these lines:
\par
\mathtoolsset{mathic} % or \mathtoolsset{mathic=true}
Subset of $(V)$ and subset of $(A)$. \par
\mathtoolsset{mathic=false}
Subset of $(V)$ and subset of $(A)$. \par
\end{quote}

Compare these lines:

Subset of $V$ and subset of $A$.

Subset of $V$ and subset of $A$.

As of 2013, $\textbackslash (\text{and} \textbackslash )$ are robust, as are the italic corrected versions.

\textbf{Caveat}: Italic correction is a treacherous area. For example any penalties will cancel the italic correction inserted by $\textbackslash$ (for an explanation see [8], section 4.3.3). We have changed Michael’s original to accommodate one specific penalty construction: the \texttt{tie}, i.e., ‘\texttt{text
\textbackslash}’ will work as expected (as of July, 2014).

4.2 Left sub/superscripts

\begin{quote}
\prescript{〈sup}{〈sub}{〈arg}\}
\prescript-sup-format = 〈cmd\}
\prescript-sub-format = 〈cmd\}
\prescript-arg-format = 〈cmd\}
\end{quote}

Sometimes one wants to put a sub- or superscript on the left of the argument. The \texttt{\prescript} command does just that:

\begin{verbatim}
\prescript{\{\textbf{C}\}}{5+}{\{12\}} \prescript{\{\textbf{C}\}}{5+}{\{14\}} \prescript{\{\textbf{C}\}}{5+}{\{12\}} \prescript{\{\textbf{C}\}}{5+}{\{14\}}
\end{verbatim}

$\prescript{\{\textbf{C}\}}{5+}{\{12\}}$ $\prescript{\{\textbf{C}\}}{5+}{\{14\}}$

The formatting of the arguments is controlled by three keys. This silly example shows you how to use them:

\begin{verbatim}
\newcommand*{\myisotope}[3]{{}\% \begingroup % to keep changes local. We cannot use a brace group % as it affects spacing!
\end{verbatim}

33
\[ A^Z X \to A-4^Z-2 Y + 4^2 \alpha \]

(Though a package like mhchem might be more suitable for this type of material.)

### 4.3 Declaring math sizes

```latex
\DeclareMathSizes\langle\text{dimen}\rangle\langle\text{dimen}\rangle\langle\text{dimen}\rangle\langle\text{dimen}\rangle
```

If you don’t know about \DeclareMathSizes, then skip the rest of this text. If you do know, then all that is needed to say is that with mathtools it is patched so that all regular dimension suffixes are now valid in the last three arguments. Thus a declaration such as

```
\DeclareMathSize{9.5dd}{9.5dd}{7.5dd}{6.5dd}
```

will now work (it doesn’t in standard \LaTeX). When this bug has been fixed in \LaTeX, this fix will be removed from mathtools.

The fix was added to the \LaTeX kernel in 2015. We will continue to provide it on older kernels.

### 4.4 Spreading equations

```latex
\begin{spreadlines}\langle\text{dimen}\rangle\langle\text{contents}\rangle\end{spreadlines}
```

The spacing between lines in a multiline math environment such as gather is governed by the dimension \jot. The spreadlines environment takes one argument denoting the value of \jot inside the environment:

```
\begin{spreadlines}{20pt}
Large spaces between the lines.
\begin{gather}
a=b\\
c=d
\end{gather}
\end{spreadlines}
Back to normal spacing.
\begin{spreadlines}
a=b\\
c=d
```

34
In a document set in `fleqn`, you might sometimes want an inner `gathered` environment that doesn’t center its lines but puts them flush left. The `lgathered` environment works just like the standard `gathered` except that it flushes its contents left:

\begin{equation}
\begin{lgathered}
  x=1, \quad x+1=2 \\
  y=2
\end{lgathered}
\end{equation}

\(x = 1, \ x + 1 = 2\) (17)

Similarly the `rgathered` puts it contents flush right.

More interesting is probably the command `newgathered`. In this example we define a `gathered` version that centers the lines and also prints a star and a number at the left of each line.

\begin{verbatim}
\newcounter{steplinecnt}
\newcommand\stepline{\stepcounter{steplinecnt}\thesteplinecnt}
\newgathered{stargathered}
  \llap{$*$}\quad\hfil\% \hfil for centering
  \hfil\% \hfil for centering
  \setcounter{steplinecnt}{0}\% reset counter
\end{verbatim}

With these definitions we can get something like this:

\begin{verbatim}
\begin{gather}
\begin{stargathered}
  x=1, \quad x+1=2 \\
  y=2
\end{stargathered}
\end{gather}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\renewgathered renews a gathered environment of course.

In all fairness it should be stated that the original concept by Michael has been extended quite a bit in `mathtools`. Only the end product of `\gathered` is the same.

### 4.6 Split fractions

\begin{itemize}
\item \texttt{\splitfrac{\langle start line \rangle}{\langle continuation \rangle}}
\item \texttt{\splitdfrac{\langle start line \rangle}{\langle continuation \rangle}}
\end{itemize}

These commands provide split fractions, e.g., multiline fractions:

\begin{verbatim}
\[
\begin{align*}
a &= \frac{\splitfrac{xy + xy + xy + xy + xy}{+ xy + xy + xy + xy}}{z} \\
\frac{\splitfrac{xy + xy + xy + xy + xy}{+ xy + xy + xy + xy}}{z} &= \frac{\splitfrac{xy + xy + xy + xy + xy}{+ xy + xy + xy + xy}}{z}
\end{align*}
\end{verbatim}

The difference between `\splitfrac` and `\splitdfrac` is that the former forces its arguments to be typeset in text-mode math, the latter does not.

**Note:** If you try to nest `\splitfrac` inside each other you may need to add `\mathstrut` to the first argument of the nested fraction to get the spacing look even. It is not added by default as often a more cramped look is desired:

\begin{verbatim}
\[
\frac{\splitfrac{\splitfrac{xy + xy + xy + xy + xy}{+ xy + xy + xy + xy}}{z} = \frac{\splitfrac{\splitfrac{xy + xy + xy + xy + xy}{+ xy + xy + xy + xy}}{z}}{z}}{z}
\end{verbatim}

36
\[
\frac{xy + xy + xy + xy + xy}{xy + xy + xy + xy + xy} + \frac{xy + xy + xy + xy + xy}{xy + xy + xy + xy + xy} + \frac{xy + xy + xy + xy + xy}{xy + xy + xy + xy + xy} = \frac{xy + xy + xy + xy + xy}{xy + xy + xy + xy + xy}
\]

Note how the line spaces aren’t even on the left hand side.

For even more control, one can use \texttt{aligned} or \texttt{gathered} instead of \texttt{\splitfrac}.
5 New additions

5.1 Variable math strut

\xmathstrut\langle depth increase \rangle \{ \langle increase \rangle \}

In typography we use \textit{struts} to ensure specific line spacing. In text we have the \texttt{\textbackslash strut} and in math \texttt{\textbackslash mathstrut}. Both have no width, but equal the height and depth of an »(« from the current text/math font and size. In math we often need to make minute adjustments in macro definitions etc. The \textit{extended} math strut \texttt{\textbackslash mathstrut} allows to \textit{increase} (decrease if negative) the math strut in two ways.

For an \langle \textit{increase} \rangle (a decimal number), say 0.1,

\texttt{\textbackslash mathstrut\{0.1\}}

will give you a new strut where 10\% of the \textit{total height of the normal math strut} (\texttt{\textbackslash mathstrut=}\texttt{\textbackslash mathstrut\{0\}}) will be added to both \textit{the height and the depth of the original strut} (thus 20\% gets added in total). On the other hand

\texttt{\textbackslash mathstrut\{0.2\}\{0.1\}}

will result in a strut where 20\% is added to the depth and 10\% is added to the height, resulting in a strut that is 30\% larger than normal.

The following example is inspired (with permission) from an example showcasing \texttt{\textbackslash mathstrut} in the upcoming third edition of \textit{The \LaTeX\ Companion}. The example is very relevant as the entries of the \texttt{\textbackslash cases(*)} environment are typeset in \textit{text style} math and thus may end up looking quite squished.\textsuperscript{23}

\[
\begin{cases*}
\frac{ x-1 }{ x-\sin x } & x>0 \\
0 & otherwise
\end{cases*} \quad \text{vs.} \quad 
\begin{cases*}
\frac{ \xmathstrut\{0.1\} x-1 }{ \xmathstrut\{0.25\} x-\sin x } & x>0 \\
\xmathstrut\{0.4\} \sqrt{ 1-x } & otherwise
\end{cases*}
\]

To showcase the optional argument, it is probably easiest to make the strut visible. Here you’ll see that \texttt{\textbackslash mathstrut} and \texttt{\textbackslash mathstrut\{0\}} is the same:

\[
\begin{cases*}
\frac{ x-1 }{ x-\sin x } & x>0 \\
0 & otherwise
\end{cases*} \quad \text{vs.} \quad 
\begin{cases*}
\frac{ \xmathstrut\{0.1\} x-1 }{ \xmathstrut\{0.25\} x-\sin x } & x>0 \\
\xmathstrut\{0.4\} \sqrt{ 1-x } & otherwise
\end{cases*}
\]

\textsuperscript{23}Which is why Frank suggested the macro (including its implementation) in the first place.
\newcommand\vfb[1]{\begingroup\fboxsep=0pt\boxed{\,#1\,}\endgroup}
\[
\]
a
\vfb{ \mathstrut } \ % normal strut
\vfb{ \xmathstrut{0} } \ % just 0 => normal strut
\vfb{ \xmathstrut{0.5} } \ % twice as large 50% + 50%
\vfb{ \xmathstrut{-0.1} } \ % negative gives something smaller
\vfb{ \xmathstrut[0.5]{0} } \ % change only the depth
a
\]
– the last box showcases a strut where we have only changed the depth of the strut. One can see \xmathstrut[0.5]{0} kind of the opposite of \smash[b]{...}, the former makes the depth larger and the latter ignores the depth.

References


