The mathtools package[*]

Morten Høgholm, Lars Madsen

2017/10/30

Abstract

The mathtools package is an extension package to amsmath. There are two things on mathtools' agenda: (1) correct various bugs/defeciencies in amsmath until these are fixed by the $\mathcal{AMS}$ 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 `multlined` environment.

Contents

---

[*]This file has version number v1.20, last revised 2017/10/30.

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. It is easiest to just contact Lars Madsen at

    `daleif@math.au.dk`

This is of course also the address to use in case of bug reports. Please put '`mathtools`' in the subject line.

Update 2013: We now makes \(\) 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

```
fixamsmath      donotfixamsmathbugs
```

The option `fixamsmath` (default) fixes two bugs in amsmath.[1] Should you for some reason not want to fix these bugs then just add the option `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.

```
allowspaces      disallowspaces
```

Sometimes amsmath gives you nasty surprises, as here where things look seemingly innocent:

```
\[
    \begin{gathered}
      [p] = 100 \\
      [v] = 200
    \end{gathered}
\]
```

Without mathtools this will result in this output:

$$= 100$$
$$[v] = 200$$

Yes, the `[p]` has been gobbled without any warning whatsoever.[2] This is hardly what you'd expect as an end user, as the desired output was probably something like this instead:

$$[p] = 100$$
$$[v] = 200$$

---

[1] See the online LaTeX bugs database http://www.latex-project.org/cgi-bin/ltxbugs2html under $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-LaTeX problem reports 3591 and 3614.

[2] amsmath thought the [p] was an optional argument, checked if it was t or b and when both tests failed, assumed it was a c.

With the option `disallowspaces` (default) mathtools disallows spaces in front of optional arguments where it could possibly cause problems just as amsmath does with `\\` inside the display environments. This includes the environments `gathered` (and also those shown in § 4.5 on page 34), `aligned`, `multlined`, and the extended `matrix`-environments (§ 3.4.1 on page 15). If you however want to preserve the more dangerous standard optional spaces, simply choose the option `allowspaces`.

3 Tools for mathematical typesetting

> `\mathtoolsset{⟨key val list⟩}`

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

```
\mathtoolsset{
  showonlyrefs,
  mathic % or mathic = true
}
```

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 `\smash`, `\llap`, and `\rlap`

> `\mathllap[⟨mathstyle⟩]{⟨math⟩}` `\mathclap[⟨mathstyle⟩]{⟨math⟩}`
> `\mathrlap[⟨mathstyle⟩]{⟨math⟩}` `\clap{⟨text⟩}`
> `\mathmbox{⟨math⟩}` `\mathmakebox[⟨width⟩][⟨pos⟩]{⟨math⟩}`

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

```
\[
  X = \sum_{1\le i\le j\le n} X_{ij}
\]
```

$$X = \sum_{1 \le i \le j \le n} X_{ij}$$

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

```
\[
```

```
    X = \sum_{\mathclap{1\le i\le j\le n}} X_{ij}
\]
```

$$X = \sum_{1 \le i \le j \le n} X_{ij}$$

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

$$\texttt{http://www.tug.org/TUGboat/Articles/tb22-4/tb72perlS.pdf}$$

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 TeX "feature."[3] The \smash macro is fixed too.

3.1.2 Forcing a cramped style

Posted on
comp.text.tex
Michael Herschorn
1992/07/21

| \cramped[⟨*mathstyle*⟩]{⟨*math*⟩} |

Let's look at another example where we have used \mathclap:

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

$$\sum_{a^2<b^2<c} \qquad \sum_{a^2<b^2<c} \tag{1}$$

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

$$\sum_{a^2<b^2<c} \qquad \sum_{a^2<b^2<c}$$

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

mathtools offers the command \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:

---

[3]The faulty reboxing procedure.

```
\[
  \cramped{x^2}                \leftrightarrow x^2    \quad
  \cramped[\scriptstyle]{x^2} \leftrightarrow {\scriptstyle x^2}
\]
```

$$x^2 \leftrightarrow x^2 \quad x^2 \leftrightarrow x^2$$

You may be surprised how often the cramped style can be beneficial yo your output. Take a look at this example:

```
\begin{quote}
  The 2005 Euro\TeX{} conference is held in Abbaye des
  Pr\'emontr\'es, 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 EuroTeX 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 causes 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 2005 EuroTeX 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.).

\crampedllap[⟨*mathstyle*⟩]{⟨*math*⟩}   \crampedclap[⟨*mathstyle*⟩]{⟨*math*⟩}
\crampedrlap[⟨*mathstyle*⟩]{⟨*math*⟩}

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

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

Of course you could just type

```
\sum_{\mathclap{\cramped{a^2<b^2<c}}}
```

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.

3.1.3 Smashing an operator

\smashoperator[⟨*pos*⟩]{⟨*operator with limits*⟩}

Above we shoved 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, l, r, or lr (default), denoting which side of the operator should be ignored (smashed).

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

$$V = \sum_{1\le i\le j\le n}^{\infty} V_{ij} \quad X = \sum_{1\le i\le j\le n}^{3456} X_{ij} \quad Y = \sum_{1\le i\le j\le n} Y_{ij} \quad Z = T_{1\le i\le j\le n} Z_{ij}$$

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

3.1.4 Adjusting limits of operators

\adjustlimits{⟨*operator₁*⟩}_{⟨*limit₁*⟩} {⟨*operator₂*⟩}_{⟨*limit₂*⟩}

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\ge n} \quad
  \text{b)} \lim_{n\to\infty} \max_{p^2\ge n} \quad
```

7

```
    \text{c)} \lim_{n\to\infty} \sup_{p^2\ge nK} \quad
    \text{d)} \limsup_{n\to\infty} \max_{p\ge n}
  \]
```

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

a) looks okay, but b) is not quite as good because the second limit ($p^2 \ge n$) is signifi-cantly 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 \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.

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

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

The use of \sb instead of _ is allowed.

3.1.5 Swapping space above $\mathcal{AMS}$ display math environments

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

```
\SwapAboveDisplaySkip
```

as the very first content within an $\mathcal{AMS}$ display math environment. It will then issue an \abovedisplayshortskip instead of the normal \abovedisplayskip.

Note it will not work with the `equation` or `multline` environments.

Here is an example of the effect

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

$$A = B$$

$$A = B$$

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 be affect the whole document, so the preamble is the best place to issue them.

3.2.1 The appearance of tags

```
\newtagform{⟨name⟩}[⟨inner_format⟩]{⟨left⟩}{⟨right⟩}
\renewtagform{⟨name⟩}[⟨inner_format⟩]{⟨left⟩}{⟨right⟩}
\usetagform{⟨name⟩}
```

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:

```
\newtagform{brackets}{[}{]}
```

Then we activate it:

```
\usetagform{brackets}
```

The result is then

$$E \neq mc^3 \tag*{[2]}$$

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

```
\newtagform{brackets2}[\textbf]{[}{]}
\usetagform{brackets2}
\begin{equation}
  E \neq m c^3
\end{equation}
```

$$E \neq mc^3 \tag*{[\textbf{3}]}$$

When you reference an equation with \eqref, the tag form in effect at the time of referencing 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 appeareance 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⟩}
```

An equation where the tag is produced with a manual `\tag*` shouldn't be referenced 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 referencing 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}
```

$$a = b \qquad\qquad\qquad\text{Q\&A}$$

See Q&A or is it better with Q&A*?*

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 setting 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.[4] 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}
```

---

[4]I recommend setting showmanualtags to true, else the whole idea of using `\tag` doesn't really make sense, does it?

```
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}
```

$$a = a \tag*{[4]}$$
$$b = b \tag*{[**]}$$

This should refer to the equation containing $a = a$: [4]. Then a switch of tag forms.

$$c = c$$
$$d = d \tag{5}$$

This should refer to the equation containing $d = d$: (5).

$$e = e$$

Back to normal.

$$f = f \tag{6}$$

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.

Feature request by
Rasmus Villemoes
2008/03/26

| \noeqref{⟨*label,label,…*⟩} |

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.

BUG 1: Unfortunately the use of the showonlyref introduce a bug within amsmath's typesetting of formula versus equation number. This bug manifest 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 depend 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 refered 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 `\shotintertext`.

The other thing to try is to use 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

`\xleftrightarrow[⟨sub⟩]{⟨sup⟩}`	`\xRightarrow[⟨sub⟩]{⟨sup⟩}`
`\xLeftarrow[⟨sub⟩]{⟨sup⟩}`	`\xLeftrightarrow[⟨sub⟩]{⟨sup⟩}`
`\xhookleftarrow[⟨sub⟩]{⟨sup⟩}`	`\xhookrightarrow[⟨sub⟩]{⟨sup⟩}`
`\xmapsto[⟨sub⟩]{⟨sup⟩}`	

Extensible arrows are part of amsmath in the form of the commands

> `\xrightarrow[⟨subscript⟩]{⟨superscript⟩}` and
> `\xleftarrow[⟨subscript⟩]{⟨superscript⟩}`

But what about extensible versions of say, `\leftrightarrow` 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

```
\[
  A \xLeftarrow[under]{over} B
\]
```

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

`\xrightharpoondown[`⟨*sub*⟩`]{`⟨*sup*⟩`}`	`\xrightharpoonup[`⟨*sub*⟩`]{`⟨*sup*⟩`}`
`\xleftharpoondown[`⟨*sub*⟩`]{`⟨*sup*⟩`}`	`\xleftharpoonup[`⟨*sub*⟩`]{`⟨*sup*⟩`}`
`\xrightleftharpoons[`⟨*sub*⟩`]{`⟨*sup*⟩`}`	`\xleftrightharpoons[`⟨*sub*⟩`]{`⟨*sup*⟩`}`

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.

$$A \xrightharpoondown[below]{above} B \qquad A \xrightharpoonup[below]{above} B$$

$$A \xleftharpoondown[below]{above} B \qquad A \xleftharpoonup[below]{above} B$$

$$A \xrightleftharpoons[below]{above} B \qquad A \xleftrightharpoons[below]{above} B$$

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*⟩`]` `[`⟨*bracket height*⟩`]{`⟨*arg*⟩`}`
`\overbracket[`⟨*rule thickness*⟩`]` `[`⟨*bracket height*⟩`]{`⟨*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:

`$\underbracket {foo\ bar}_{baz}$` $\underbracket{foo\ bar}_{baz}$
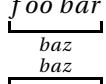
`$\overbracket {foo\ bar}^{baz}$` $\overbracket{foo\ bar}^{baz}$
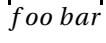
The default rule thickness is equal to that of `\underbrace` (app. 5/18 ex) while the default bracket height is equal to app. 0.7 ex. These values give really pleasing results

in all font sizes, but feel free to use the optional arguments. That way you may get "beauties" like

```
\[
  \underbracket[3pt]{xxx\  yyy}_{zzz} \quad \text{and} \quad
  \underbracket[1pt][7pt]{xxx\  yyy}_{zzz}
\]
```

$$\underbracket{xxx\ yyy}_{zzz} \quad \text{and} \quad \underbracket{xxx\ yyy}_{zzz}$$

\underbrace{⟨*arg*⟩}	\LaTeXunderbrace{⟨*arg*⟩}
\overbrace{⟨*arg*⟩}	\LaTeXoverbrace{⟨*arg*⟩}

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.

$$\sum_n \underbrace{foof}_{zzz}$$

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

$$\sum_n \underbrace{foof}_{zzz}$$

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

$$\sum_n \underbrace{foof}_{zzz}$$

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 fixes the spacing issues and apart from working with the default Computer Modern fonts, they also work with the packages mathpazo, pamath, fourier, eulervm, cmbright, and mathptmx. If you use the ccfonts to get the full Concrete fonts, the original version saved under the names \LaTeXunderbrace and \LaTeXoverbrace are better, due to of 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

```
\begin{matrix*} [⟨col⟩] ⟨contents⟩ \end{matrix*}
\begin{pmatrix*}[⟨col⟩] ⟨contents⟩ \end{pmatrix*}
\begin{bmatrix*}[⟨col⟩] ⟨contents⟩ \end{bmatrix*}
\begin{Bmatrix*}[⟨col⟩] ⟨contents⟩ \end{Bmatrix*}
\begin{vmatrix*}[⟨col⟩] ⟨contents⟩ \end{vmatrix*}
\begin{Vmatrix*}[⟨col⟩] ⟨contents⟩ \end{Vmatrix*}
```

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 orig-
inal 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 environ-
ment, 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{smallmatrix*} [⟨col⟩] ⟨contents⟩ \end{smallmatrix*}
\begin{psmallmatrix} ⟨contents⟩ \end{psmallmatrix}
\begin{psmallmatrix*}[⟨col⟩] ⟨contents⟩ \end{psmallmatrix*}
\begin{bsmallmatrix} ⟨contents⟩ \end{bsmallmatrix}
\begin{bsmallmatrix*}[⟨col⟩] ⟨contents⟩ \end{bsmallmatrix*}
\begin{Bsmallmatrix} ⟨contents⟩ \end{Bsmallmatrix}
\begin{Bsmallmatrix*}[⟨col⟩] ⟨contents⟩ \end{Bsmallmatrix*}
\begin{vsmallmatrix} ⟨contents⟩ \end{vsmallmatrix}
\begin{vsmallmatrix*}[⟨col⟩] ⟨contents⟩ \end{vsmallmatrix*}
\begin{Vsmallmatrix} ⟨contents⟩ \end{Vsmallmatrix}
\begin{Vsmallmatrix*}[⟨col⟩] ⟨contents⟩ \end{Vsmallmatrix*}
smallmatrix-align = ⟨c,l or r⟩
smallmatrix-inner-space = \,
```

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

yields

$$\begin{bmatrix} a & -b \\ -c & d \end{bmatrix} \begin{bmatrix} a & -b \\ -c & d \end{bmatrix}$$

Inside the `Xsmallmatrix` construction a small space is inserted between the fences and the contents, the size of it can be changed using smallmatrix-align=⟨*some spacing command*⟩, the default is \,.

As an extra trick the fences will behave as open and closing fences in constract to their auto-scaling nature.[5]

3.4.2 The `multlined` **environment**

```
\begin{multlined}[⟨pos⟩][⟨width⟩] ⟨contents⟩ \end{multlined}
\shoveleft[⟨dimen⟩]{⟨arg⟩}   \shoveright[⟨dimen⟩]{⟨arg⟩}
firstline-afterskip = ⟨dimen⟩     lastline-preskip = ⟨dimen⟩
multlined-width = ⟨dimen⟩        multlined-pos = c|b|t
```

Some of the amsmath environments exist in two forms: an outer and an inner environment. One example is the pair `gather` & `gathered`. There is one important omission on this list however, as there is no inner `multlined` environment, so this is where 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 `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 firstline-afterskip key. The last line should be set flush right and preceded by a hard space of size lastline-preskip. Both these hard spaces

---

[5]\left and \right do *not* produce open and closing fences, thus the space before or after may be too large. Inside this construction they behave.

have a default value of `\multlinegap`. Here we use a 't' in the first optional argument
denoting a top-aligned building block (the default is 'c').

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

$$A = \boxed{\text{first}} \quad B$$
$$\boxed{\text{last}}$$

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

```
\[
  \begin{multlined}
    \framebox[.65\columnwidth]{First line}        \\
    \framebox[.5\columnwidth]{Second line}         \\
    \shoveleft{L+E+F+T}                            \\
    \shoveright{R+I+G+H+T}                         \\
    \shoveleft[1cm]{L+E+F+T}                       \\
    \shoveright[\widthof{$R+I+G+H+T$}]{R+I+G+H+T} \\
    \framebox[.65\columnwidth]{Last line}
  \end{multlined}
\]
```

$$\boxed{\text{First line}}$$
$$\boxed{\text{Second line}}$$
$$L + E + F + T$$
$$R + I + G + H + T$$
$$L + E + F + T$$
$$R + I + G + H + T$$
$$\boxed{\text{Last line}}$$

You can also choose the width yourself by specifying it as an optional argument:

```
\[
  \begin{multlined}[b][7cm]
    \framebox[4cm]{first} \\
    \framebox[4cm]{last}
  \end{multlined} = B
\]
```

$$\boxed{\text{first}}$$
$$\boxed{\text{last}} = B$$

There can be two optional arguments (position and width) and they're interchange-
able.

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 `multline` row, inside an `array` this line is no longer 'invisible' because `array` sets `\baselineskip` to zero. Currently we have no general workaround for this.

Bug 2: Due to the way `multilined` is implemented, certain constructions does not work inside `multlined`. We have added a hook (`\MultlinedHook`) that can be added to. The default value is a fix for `subarray` and thus `\substack`.

This can actually be used to fix Bug 1:

```
\usepackage{mathtools,etoolbox}
\newlength\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

```
\begin{dcases}  ⟨math_column⟩ & ⟨math_column⟩ \end{dcases}
\begin{dcases*} ⟨math_column⟩ & ⟨text_column⟩  \end{dcases*}
\begin{rcases}    ⟨math_column⟩ & ⟨math_column⟩ \end{rcases}
\begin{rcases*}   ⟨math_column⟩ & ⟨text_column⟩  \end{rcases*}
\begin{drcases}   ⟨math_column⟩ & ⟨math_column⟩ \end{drcases}
\begin{drcases*}  ⟨math_column⟩ & ⟨text_column⟩  \end{drcases*}
\begin{cases*}  ⟨math_column⟩ & ⟨text_column⟩  \end{cases*}
```

Anyone who have tried to use an integral in the regular `cases` environment from amsmath 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.

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

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

Additionally the environment `dcases*` acts just the same, but the second column is set in the normal roman font of the document.[6]

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

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

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.

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

$$\left. \begin{array}{ll} x^2 & \text{for } x > 0 \\ x^3 & \text{else} \end{array} \right\} \quad \Rightarrow \cdots$$

3.4.4 Emulating indented lines in alignments

`\MoveEqLeft[⟨number⟩]`

In [7], Ellen Swanson recommends that when ever 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 displayes that end up looking like this

$$\boxed{\text{Long first line}}$$
$$= \boxed{\text{2nd line}}$$
$$\leq \ldots$$

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

```
\begin{align*}
  \MoveEqLeft \framebox[10cm][c]{Long first line}\\
  & = \framebox[6cm][c]{ \hphantom{g} 2nd line}\\
```

---

[6]Or rather: it inherits the font characteristics active just before the `dcases*` environment.

```
        & \leq \dots
    \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] \framebox[10cm][c]{Part 1}\\
    = {} & \framebox[8cm][c]{2nd line}\\
        & + \framebox[4cm][c]{ last part}
\end{align*}
```

	Long first line

		2nd line
=		

		last part
+		
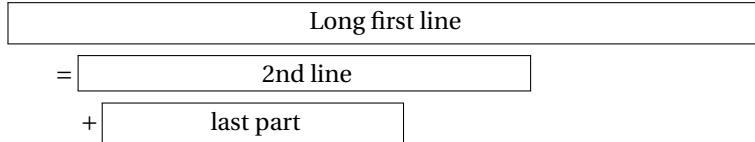
3.4.5 Boxing a single line in an alignment

The amsmath package provie 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.[7]

\Aboxed{⟨*left hand side*   &   *right hand side*⟩}

Example

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

Resulting in:

$$\boxed{f(x) = \int h(x)\, dx}$$
$$= g(x)$$

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*}
```

---

[7]Note that internally \Aboxed does use \boxed.

20

$$\boxed{f(x) = 0} \qquad\qquad \boxed{g(x) = b}$$
$$\boxed{h(x)} \qquad\qquad\qquad \boxed{i(x)}$$

Note how the `\fboxsep` change only affect 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.

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.

```
\ArrowBetweenLines[⟨symbol⟩]
\ArrowBetweenLines*[⟨symbol⟩]
```

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}
  && \framebox[1.5cm]{} &= \framebox[3cm]{}\\
  \ArrowBetweenLines % \Updownarrow is the default
  && \framebox[1.5cm]{} &= \framebox[2cm]{}
\end{alignat}
```

resulting in

$$\boxed{} = \boxed{} \tag{7}$$
$$\Updownarrow$$
$$\boxed{} = \boxed{} \tag{8}$$

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

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

resulting in

$$\boxed{} = \boxed{}$$
$$\Downarrow$$
$$\boxed{} = \boxed{}$$

21

3.4.7 Centered `\vdots`

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

$$\boxed{} = \boxed{}$$
$$\vdots$$
$$= \boxed{}$$

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.

```
\vdotswithin{⟨symbol⟩}
\shortvdotswithin{⟨symbol⟩}
\shortvdotswithin*{⟨symbol⟩}
\MTFlushSpaceAbove
\MTFlushSpaceBelow
shortvdotsadjustabove = ⟨length⟩
shortvdotsadjustbelow = ⟨length⟩
```

Two examples in one

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

yielding

$$a = b$$
$$\vdots$$
$$= c$$
$$\vdots$$
$$= d$$

Thus `\vdotswithin{=}` create a box corersponding to {}={} and typeset a »$\vdots$« 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.

 `\shortvdotswithin{=}` corresponds to

```
\MTFlushSpaceAbove
& \vdotswithin{=} \\
\MTFlushSpaceBelow
```

22

whereas `\shortvdotswithin*{=}` is the case with `\vdotswithin{=} & \\`. This also means one cannot write more on the line when using `\shortvdotswithin` or the starred version. But one can de-construct the macro and arrive at

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

yielding

$$
\begin{aligned}
A + B &= C + D \\
\vdots \qquad &\quad \vdots \\
C + D &= Y + K
\end{aligned}
$$

If one has the need for such a construction.

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

3.5 Intertext and short intertext

| `\shortintertext{⟨text⟩}` |

amsmath provides the command `\intertext` for interrupting a multiline display while still maintaining the alignment points. However the spacing often seems quite excessive as seen below.

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

$$a = b \tag{9}$$

Some text

$$c = d \tag{10}$$

Using the command `\shortintertext` alleviates this situation somewhat:

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

$$a = b \tag{11}$$

Some text

$$c = d \tag{12}$$

It turns out that both \shortintertext and the original \intertext from amsmath has a slight problem. If we use the 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 \shortintertext and \intertext. It can be illustrated using the following example, an interested reader, can apply it with and with out 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 internaly for both \intertext and \shortintertext, plus we add the posibility to fine tune spacing around these constructions. The original versions can be brought back using the original-x keys below.

\intertext{⟨*text*⟩}
\shortintertext{⟨*text*⟩}
original-intertext = true|false   (default: false)
original-shortintertext = true|false   (default: false)
above-intertext-sep = ⟨*dimen*⟩   (default: 0pt)
below-intertext-sep = ⟨*dimen*⟩   (default: 0pt)
above-shortintertext-sep = ⟨*dimen*⟩   (default: 3pt)
below-shortintertext-sep = ⟨*dimen*⟩   (default: 3pt)

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

```
\DeclarePairedDelimiter{⟨cmd⟩}{⟨left_delim⟩}{⟨right_delim⟩}
```

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

```
\newcommand*\abs[1]{\lvert#1\rvert}
\newcommand*\norm[1]{\lVert#1\rVert}
```

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 `\left`–`\right` pair or a `\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.:

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

  $$\left|\frac{a}{b}\right|$$

- A variant with an optional argument: `\abs[⟨size_cmd⟩]`, where ⟨*size_cmd*⟩ is either `\big`, `\Big`, `\bigg`, or `\Bigg` (if you have any bigggger versions you can use them too).

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

  $$\Bigg|\frac{a}{b}\Bigg|$$

```
\DeclarePairedDelimiterX{⟨cmd⟩}[⟨num args⟩]{⟨left_delim⟩}{⟨right_delim⟩}{⟨body⟩}
\delimsize
```

Sometimes one might want to have the capabilities of `\DeclarePairedDelimiter`, but also want a macro the takes more than one argument and specify 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 ⟨*body*⟩ for what ever the macro should do within these fences.

Inside the ⟨*body*⟩ part, the macro `\delimsize` refer to the size of the outer fences. It can then be used inside ⟨*body*⟩ to scale any inner fences.

Thus

```
\DeclarePairedDelimiter{⟨cmd⟩}{⟨left_delim⟩}{⟨right_delim⟩}
```

is the same thing as

```
\DeclarePairedDelimiterX{⟨cmd⟩}[1]{⟨left_delim⟩}{⟨right_delim⟩}{#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 ⟨*code*⟩. Here we do a weird three argument 'braket'

```
\DeclarePairedDelimiterX\braket[3]{\langle}{\rangle}%
{#1\,\delimsize\vert\,\mathopen{}#2\,\delimsize\vert\,\mathopen{}#3}
```

Note the use of »`\mathopen{}`*« in the ⟨body⟩ 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}` *in a version without* `\mathopen{}`*.*[8]

Then we get

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

$$\left\langle A, \frac{1}{2} \right\rangle \quad \left\langle B \left| \sum_k \right| C \right\rangle$$

---

[8]Basically, the problem is that `\vert` is a 'symbol', thus $|-B$ is interpreted *subtraction*, not a symbol followed by negative B. When `\mathopen{}` is added, TeX is told that an opening delimiter just occurred, and it will adjust the minus accordingly.

With the inner scaling, we can provide macros whos syntax closely follow the mathematical meaning. Fx for building sets, try this[9]

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

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

$$\left\{ x \in X \,\middle|\, \frac{\sqrt{x}}{x^2+1} > 1 \right\}$$

Thus we end up with a syntax much closer to how we read this aloud. Also we hide the 'given' symbol for easy replacement.

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

```
\usepackage{etoolbox}
\DeclarePairedDelimiterX\norm[1]\lVert\rVert{
    \ifblank{#1}{\:\cdot\:}{#1}
}
```

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

\DeclarePairedDelimiterXPP{⟨*cmd*⟩}[⟨*num args*⟩]{⟨*pre code*⟩}{⟨*left_delim*⟩}
{⟨*right_delim*⟩}{⟨*post code*⟩}{⟨*body*⟩}

\DeclarePairedDelimiterX has an annoying caveat: it is very hard to make a macro \lnorm{a} that should result in $\|a\|_2$.[10]

As a consequence we provide \DeclarePairedDelimiterXPP.[11] With the addition of the {⟨*pre code*⟩} and {⟨*post code*⟩} it is identical to \DeclarePairedDelimiterX. It should be interpreted as

{⟨*pre code*⟩} {⟨*left_delim*⟩} {⟨*body*⟩} {⟨*right_delim*⟩} {⟨*post code*⟩}

---

[9]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.

[10]The added »_2« is hard to get around the argument parsing.

[11]Extended DeclarePairedDelimiter with Pre and Post code.

An $\mathscr{L}^2$ norm can now be defined as

```
\DeclarePairedDelimiterXPP\lnorm[1]{}{\lVert\rVert{_2}{#1}
```

A probability macro with build in support for conditionals (\given initialized as above)

```
\DeclarePairedDelimiterXPP\Prop[1]{\mathbb{P}}(){}{
  \renewcommand\given{\nonscript\:\delimsize\vert\nonscript\:\mathopen{}}
  #1}
```

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

Note 1: As the number of arguments increase the \DeclarePairedDelimiter... macros become hard for users to understand. A key-value interface would be better. This is planed for a future release. In `http://tex.stackexchange.com/a/136767/` `3929` there is a suggested replacement for \DeclarePairedDelimiter, that greatly reduces the number of macros and provides a key-val interface. However, the code use 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 [3] for the number of arguments. We will consider this for a future release.

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

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 gab between 'sin' and '('. In some special cases it may be useful to be able to tinker with the behavior.

\reDeclarePairedDelimiterInnerWrapper{⟨*macro name*⟩}{⟨star *or* nostarnonscaled *or* nostarscaled⟩}{⟨*code*⟩}

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 ⟨*code*⟩, #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:

```
\DeclarePairedDelimiter\abs\lvert\rvert
\reDeclarePairedDelimiterInnerWrapper\abs{star}{#1#2#3}
\reDeclarePairedDelimiterInnerWrapper\abs{nostarnonscaled}{\mathinner{#1#2#3}}
\reDeclarePairedDelimiterInnerWrapper\abs{nostarscaled}{\mathinner{#1#2#3}}
```

The default values for the call backs corresponds to

```
star:            \mathopen{}\mathclose\bgroup #1#2\aftergroup\egroup #3
nostarnonscaled: \mathopen#1#2\mathclose#3
nostarscaled:    \mathopen{#1}#2\mathclose{#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).[12]

```
\mathclose{\rvert}^2\mathclose\rvert^2\rvert^2
```

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 is typeset inside a group (to limit `\delimsize`) and thus hide any `&` or `\\` from `align` and `align` breaks down.

<div style="margin-left:0">Feature provided by Sebastien Gouezel 2014/05/14</div>

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}[t]
   \abs*{ & \frac{1}{2}+\cdots \\
         & \dots+\frac{1}{2} }
 \end{aligned}
\end{align*}
```

---

[12]Interestingly it dod not show up in the font of this manual, which uses the fourier font set.

$$f(a) = \left| \frac{1}{2} + \cdots \right.$$

$$\left. \cdots + \frac{1}{2} \right|$$

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]
```

LATEX 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[13] so mathtools defines the commands `\lparen` and `\rparen` to represent the left and right parenthesis resp.

3.7.2 Vertically centered colon

centercolon = true|false
`\vcentcolon`   `\ordinarycolon`

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 `\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: $a := b$. for now

Typing `\vcentcolon` every time is quite tedious, so one can use the key centercolon to make the colon active instead.

```
\mathtoolsset{centercolon}
\[
  a := b
\]
\mathtoolsset{centercolon=false}
```

$$a := b$$

In this case the command `\ordinarycolon` typesets an ... ordinary colon (what a surprise).

---

[13]The empheq package needs command names for delimiters in order to make auto-scaling versions.

Warning: `centercolon` *does not* work with languages that make use of an active colon, most notably *French*. Sadly the `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.

```
\coloneqq      \Coloneqq      \coloneq    \Coloneq
\eqqcolon      \Eqqcolon      \eqcolon    \Eqcolon
\colonapprox   \Colonapprox   \colonsim   \Colonsim
\dblcolon
```

The font packages txfonts and pxfonts provides various symbols that include a vertically centered colon but with tighter spacing. For example, the combination `:=` exists as the symbol `\coloneqq` which typesets as $\coloneqq$ instead of $:=$. The primary disadvantage of using these fonts are the support packages' lack of support for amsmath (and thus mathtools) and worse yet, the side-bearings are way too tight; see [4] for examples. If you're not using these fonts, mathtools provides the symbols for you. Here are a few examples:

```
\[
  a \coloneqq b \quad c \Colonapprox d \quad e \dblcolon f
\]
```

$$ a \coloneqq b \quad c \Colonapprox d \quad e \dblcolon f $$

3.7.3 A few missing 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.

```
\nuparrow
\ndownarrow
\bigtimes
```

Note: that 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 needs 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

```
\[
  \lim_{a\ndownarrow 0} f(a) \neq \bigtimes_n X_n \qquad
  \frac{ \bigtimes_{k=1}^7 B_k \nuparrow \Omega }{2}
\]
```

$$ \lim_{a \ndownarrow 0} f(a) \neq \bigtimes_n X_n \qquad \frac{\bigtimes_{k=1}^7 B_k \nuparrow \Omega}{2} $$

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 TeX 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

mathic = true|false

TeX usually takes care of italic corrections in text, but fails when it comes to math. If you use the LaTeX inline math commands \(and \) you can however work around it by setting the key 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, \(\) are robust, as is the italic corrected versions.

Caveat: Italic correction is a treacherous area. For example any penalties will cancel the italic correction inserted by \((for an explanation see [8], section 4.3.3). We have changed Michaels original to accomodate one specific penalty construction: the *tie*, i.e., »text~\(« will work as expected (as of July, 2014).

4.2 Left sub/superscripts

\prescript{⟨*sup*⟩}{⟨*sub*⟩}{⟨*arg*⟩}	prescript-sup-format = ⟨*cmd*⟩
prescript-sub-format = ⟨*cmd*⟩	prescript-arg-format = ⟨*cmd*⟩

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

```
\[
  {}^{4}_{12}\mathbf{C}^{5+}_{2}                \quad
  \prescript{14}{2}{\mathbf{C}}^{5+}_{2}   \quad
  \prescript{4}{12}{\mathbf{C}}^{5+}_{2}   \quad
  \prescript{14}{}{\mathbf{C}}^{5+}_{2}    \quad
  \prescript{}{2}{\mathbf{C}}^{5+}_{2}
\]
```

$$\prescript{4}{12}{\mathbf{C}}^{5+}_{2} \quad \prescript{14}{2}{\mathbf{C}}^{5+}_{2} \quad \prescript{4}{12}{\mathbf{C}}^{5+}_{2} \quad \prescript{14}{}{\mathbf{C}}^{5+}_{2} \quad \prescript{}{2}{\mathbf{C}}^{5+}_{2}$$

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

```
\newcommand*\myisotope[3]{%
  \begingroup % to keep changes local. We cannot use a brace group
             % as it affects spacing!
    \mathtoolsset{
      prescript-sup-format=\mathit,
      prescript-sub-format=\mathbf,
      prescript-arg-format=\mathrm,
    }%
  \prescript{#1}{#2}{#3}%
  \endgroup
}
\[
  \myisotope{A}{Z}{X}\to \myisotope{A-4}{Z-2}{Y}+
  \myisotope{4}{2}{\alpha}
\]
```

$$\prescript{A}{Z}{\mathrm{X}} \to \prescript{A-4}{Z-2}{\mathrm{Y}} + \prescript{4}{2}{\alpha}$$

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

4.3 Declaring math sizes

$\boxed{\texttt{\\DeclareMathSizes\{}\langle dimen\rangle\texttt{\}\{}\langle dimen\rangle\texttt{\}\{}\langle dimen\rangle\texttt{\}\{}\langle dimen\rangle\texttt{\}}}$

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 i 2015. We will continue to provide it on older kernels.

4.4   Spreading equations

> \begin{spreadlines}{⟨*dimen*⟩} ⟨*contents*⟩ \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{gather}
  a=b\\
  c=d
\end{gather}
```

Large spaces between the lines.

$$a = b \tag{13}$$

$$c = d \tag{14}$$

Back to normal spacing.

$$a = b \tag{15}$$
$$c = d \tag{16}$$

4.5   Gathered environments

> \begin{lgathered}[⟨*pos*⟩] ⟨*contents*⟩ \end{lgathered}
> \begin{rgathered}[⟨*pos*⟩] ⟨*contents*⟩ \end{rgathered}
> \newgathered{⟨*name*⟩}{⟨*pre_line*⟩}{⟨*post_line*⟩}{⟨*after*⟩}
> \renewgathered{⟨*name*⟩}{⟨*pre_line*⟩}{⟨*post_line*⟩}{⟨*after*⟩}

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
```

34

```
  \end{lgathered}
 \end{equation}
```

$$x = 1, \quad x + 1 = 2$$
$$y = 2$$

(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.

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

With these definitions we can get something like this:

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

$$1* \quad x = 1, \quad x + 1 = 2$$
$$2* \qquad y = 2$$

(18)

`\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 `lgathered` is the same.

4.6   Split fractions

`\splitfrac{⟨numer⟩}{⟨denom⟩}`   `\splitdfrac{⟨numer⟩}{⟨denom⟩}`

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

```
\[
  a=\frac{
      \splitfrac{xy + xy + xy + xy + xy}
               {+ xy + xy + xy + xy}
    }
    {z}
  =\frac{
      \splitdfrac{xy + xy + xy + xy + xy}
                {+ xy + xy + xy + xy}
```

```
        }
        {z}
    \]
```

$$a = \frac{\begin{matrix} xy + xy + xy + xy + xy \\ + xy + xy + xy + xy \end{matrix}}{z} = \frac{\begin{matrix} xy + xy + xy + xy + xy \\ + xy + xy + xy + xy \end{matrix}}{z}$$

References

[1] Alexander R. Perlis, *A complement to `\smash, \llap, and \rlap`*, TUGboat 22(4) (2001).

[2] American Mathematical Society and Michael Downes, *Technical notes on the amsmath package* Version 2.0, 1999/10/29. (Available from CTAN as file `technote.tex`.)

[3] Frank Mittelbach, Rainer Schöpf, Michael Downes, and David M. Jones, *The amsmath package* Version 2.13, 2000/07/18. (Available from CTAN as file `amsmath.dtx`.)

[4] Frank Mittelbach and Michel Goossens. *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Boston, Massachusetts, 2 edition, 2004. With Johannes Braams, David Carlisle, and Chris Rowley.

[5] David Carlisle, *The keyval Package*, Version 1.13, 1999/03/16. (Available from CTAN as file `keyval.dtx`.)

[6] Herbert Voß, *Math mode*, Version 1.71, 2004/07/06. (Available from CTAN as file `Voss-Mathmode.pdf`.)

[7] Ellen Swanson, *Mathematics into type*. American Mathematical Society, updated edition, 1999. Updated by Arlene O'Sean and Antoinette Schleyer.

[8] Victor Eijkhout, *TeX by Topic, A TexnicianâĂŹs Reference*, 2007. Freely available at `http://ctan.org/tex-archive/info/texbytopic`.