autonum — automatic number referenced equations*

Patrick Häcker†

Released 2015/01/18

1 Introduction

With LaTeX, the user has to decide manually to not number an equation by adding a star in the math environment. Authors who do not want to think about the numbering during writing (and maybe they should not), often use the environments without stars. As default, these users get all equations numbered, although there are different opinions about what should be numbered.

The other automatic possibility one can think of is to number only the referenced equations. The mathtools package's option `showonlyrefs` seems to be the solution for those who want to have the referenced equations numbered only. Unfortunately, for amsmath users this also means a step backwards, as the numbering can overwrite parts of the equation according to its documentation. Generally, this options seems to be quite unreliable as it is incompatible with the cleveref package. The autonum package implements the numbering of referenced equations only without this deficiencies.

2 Usage and Examples

To get the automatic numbering of referenced equations, simply load the package:

```
\usepackage{autonum}
```

This provides support for `\ref` (and `\cref` if cleveref is loaded, see section 3.1). If you need support for other reference commands, see section 3.3. As you probably use the hyperref package, you should set the hypperref option `hypertexnames=false` when loading the hyperref package to avoid some warnings (see section 3.2).

The recommended style is to add a label to each logical equation. Then, simply use the references as needed. Never use the starred forms when defining an equation as they do not make sense with autonum and are deactivated. You

*This file describes version v0.3.11, last revised 2015/01/18.

†E-mail: pat_h@web.de

can use the (redefined) `equation` or `\[-\]`-environment in most cases, if you add the appropriate `\begin{split}-\end{split}` pairs when needed. Sometimes, an `align`, `multline`, `gather`, `flalign` or `alignat` environment is needed. Do not use `eqnarray`.

Please keep in mind, that using autonum might not always be a good thing. If you hand-in a paper for review with many equations on a page, you might avoid using autonum or if you do, you probably should reference most of your equations or activate line numbers.

`equation` The following examples show the results of the equation environment after loading the package. Now let's reference the third equation: 1.

$$1 = 1 \quad \text{(equation without label)}$$

$$2 = 2 \quad \text{(equation with label, not referenced)}$$

$$3 = 3 \quad \text{(equation with label, already referenced)} \tag{1}$$

$$4 = 4 \quad \text{(equation with label, referenced later)} \tag{2}$$

Now let's reference the fourth equation: 2. The first and the second equations do not get a number, as they are not referenced.

If you want to try an example yourself, you can take this one, where only the first equation gets a number:

```
\documentclass{article}
\usepackage{autonum}
\begin{document}
   \begin{equation}\label{a}
      a
   \end{equation}
   \begin{equation}\label{b}
      b
   \end{equation}
   \ref{a}
\end{document}
```

To get the correct references up to three compilation runs are necessary when the autonum package is used (as always in TEX this cannot be guaranteed, so in rare cases even that is not enough). This is one additional run compared to the default referencing mode, as one run is needed to check if an equation is used. This is probably not avoidable, as the information which equation should have a number is simply not always available in the first run while layouting the equation. Thus, the equation's number can change in the second run. For the reference command, this information is only stable in the third run, as the equations influence each other.

`align et al.` Instead of equation you may also use one of the following environments: `align`, `gather`, `multline`, `flalign`. The label should be given after the equation but before the newline command, e.g.

```
\begin{align}
   a\label{firstLabel}\\
   b\label{secondLabel}
\end{align}
```

Providing the label at the beginning of an equation's line should work, too, but is less thoroughly tested. Please provide feedback, if you find a non-working constellation, which works if autonum is not loaded.

`split` The split environment should work as expected with one exception. To get correct vertical spacing after a non-referenced split environment, which is embedded into another math environment, autonum must execute code inside of the split environment. To identify the environment, currently the label name is used, which is not available inside of the split environment, if the label is given after the end of the split environment, which is possible without autonum. So as an exception to the rule above, provide the label before the beginning of the split environment when using a split environment to be on the safe side. Providing it inside of the split environment seems to work currently, but is not guaranteed to work in the future. If the label is given too late, autonum prints an error to avoid accidently introducing a spacing or numbering problem. As an example, put the labels as following:

```
\begin{align}
   \label{firstLabel}
   \begin{split}
      a
      \\
      b
   \end{split}
   \\
   c\label{secondLabel}
\end{align}
```

`\[ and \]` As the commands `\[` and `\]` are useless when using autonum, they are redefined to be an alias of `begin{equation}` and `end{equation}`, respectively. Without adding labels or referencing added labels, the new `\[-\]`-environment behaves like the old one.

`equation+` In the rare case, that an equation is very important and not references within the text, but some other person wants to reference to that equation, you can use `\begin{equation+}` and `\end{equation+}`. The equation is then numbered in all cases (having a label or not, being referenced or not). This feature has been

added, so that all three referencing practices are supported in LaTeX with as much automation as possible. The other math environments support a `+`-variant, too.

`equation*` The starred versions have been deleted, to avoid accidental use. Use the versions without star and without adding a label, to avoid that an equation gets a number.

3 Compatibility

3.1 Load Order

As other packages might break autonum, it should be loaded very late. Normally, amsmath is loaded automatically to make use of the more advanced math environments. Nevertheless, to use other packages, it might be necessary to load amsmath manually. To use autonum with amsmath, hyperref and cleveref for example, the order must be amsmath → hyperref → cleveref → autonum. If a wrong order has been active, it might be necessary to delete the aux file manually to get rid of compile errors, which is generally a good debugging strategy if autonum is or was involved. The support of autonum without amsmath has been discontinued due to missing demand.

3.2 Hyperref

The hyperref package must be loaded with the option `hypertexnames=false` to work when autonum is used, as in the following example

```
\usepackage[hypertexnames=false]{hyperref}
\usepackage{autonum}
```

Problem description: Generally, the autonum package is careful not to break other packages, but with autonum there are non-starred mathematical environment which do not increase the equation counter if they are not referenced. This leads to the following warning in hyperref when more than one equation is used: "destination with the same identifier (nameequation.0.1) has been already used, duplicate ignored".

Additionally, the link anchors do not work correctly (with a reference followed by a labeled equation, there is a warning in tracing mode, too. Hyperref seems to increase LaTeX's equation's counter (called `equation`) only if a equation is referenced. This counter may not be modified to avoid problems with Hyperref, as this would change the displayed equation number (and it does not work anyway). This problem is independent of cleveref. The problem might be solvable by modifying `\begin{equation}` or `\end{equation}`. Using `gather` instead of `equation` removes the warning, but hyperlinks still do not work.

So it's a mess and probably not worth the time, as `hypertexnames=false` solves all these problems.

3.3 Reference commands

As default the `\ref` and cleveref's `\cref` commands are supported (the latter one only if the cleveref package is loaded). If you want to add autonum support for other referencing commands, you have the following choices. Please mind that these interfaces might change in the future, as it's unclear what reference commands make sense beyond `\cref`. Please also mind the `\makeatletter` and `\makeatother` command before and after the commands, respectively.

`utonum@generatePatchedReference` For normal reference commands expecting a single label name as an argument, you can use `\autonum@generatePatchedReference{NewReferenceCommand}`.

`num@generatePatchedReferenceCSL` For reference commands expecting a comma separated list of label names as an argument, you can use
`\autonum@generatePatchedReferenceCSL{NewReferenceCommand}`.

`eneratePatchedReferenceGeneral` For reference commands expecting a general data structure containing label names as an argument, you can use
`\autonum@generatePatchedReferenceGeneral{NewReferenceCommand}{\SplitAndLoopMacro}`.
The macro `\SplitAndLoopMacro` acts as a function expecting the to-be-called function as the first argument and the data structure as the second argument. It must call the function given in the first argument for each label name given in the data structure of the second argument. An example is the `\forcsvlist` macro from the etoolbox package, which is used to implement `\autonum@generatePatchedReferenceCSL`.

`\crefrange` Range commands like the `\crefrange` command are not supported at the moment. It's not completely clear, if this is a good or a bad thing. Please contact the package author if you are interested in this feature.

Support for new reference command names can be added by executing
`\let\ref\NewReferenceCommand`.

If the new reference command can have optional arguments, you should probably write instead
`\LetLtxMacro\ref\NewReferenceCommand`.

If you want to use the power of `\cref` without getting rid of the habit of writing `\ref`, you can just add the following code to the preamble

```
\AfterEndPreamble{%
   \let\ref\cref
}%
```

If you also strive for maximum consistency and want to avoid accidently writing `\cref` or `\autoref`, you can instead add the following lines

```
\AfterEndPreamble{%
   \let\ref\cref
   \undef{\cref}%
```

```
  \undef{\autoref}%
}%
```

4 Further Ideas

- For equations with multiple lines, which have a width close to the linewidth, but whose width at the middle/end is not so large, the equation number should not be set below the equation but next to the equation as it is done if the equation is smaller.
- A similar approach could be used for figure and table environments to print a warning, if such an environment is not referenced. With the subfig package, the solution would be harder, as a figure or table may be unreferenced, if all subfloats are referenced. Similarly, a subfloat may be unreferenced, if its parent environment is referenced. So the warning should only be printed if an unreferenced parent environment either does not have any child environment or if there is an unreferenced child environment.
- A warning could be printed, if another compile is necessary. Sometimes this is the case, but not always.
- A "*" could be used instead of a "+".
- If a reference is used before the label is defined, the reference information is saved in a variable and can be used later in the current run when processing the label. It does not have to be saved to the aux file. If a reference is used after defining the label, the reference information is saved in the aux file and can be used in the next run when processing the label. The information does not have to be saved into a variable, as a label must only be defined once and the definition has already happened. Unfortunately, this would make it necessary to distinguish between definitions in the current and in the last run, as otherwise this leads either to oscillation or to defining everything in the end (depending if csdefaux or csdefall is used in the not-defined case), which is probably not worth the effort.
- The command `\(` could be an alias for `begin{split}` and `\)` could be an alias for `end{split}` inside of another math environment, as the LaTeX math inline syntax can only occur outside of a math environment.
- This trick might be handy
- The `\crefrange` might be supported. As this would require a lot of work it will only be done if multiple people show interest and there are really convincing real-world examples where using `\crefrange` is superior to using `\cref`. Patches are, of course, welcome, too.
- This documentation should be vastly improved from someone who is actually good in writing documentations.

5 Contributions

- David Carlisle
  - explained that amsmath environments are executed twice what lead to the support of the amsmath environments
  - created the `\vanishprotect` macro what avoided errors after the deactivation of the package
- Fg Nu
  - highlighted, that special characters in label names must be supported what lead to the support of special characters
- Joseph Wright
  - created a correctly working `\csxdefaux` what lead to the support of special characters
  - sent an MWE showing a regression in the support of special characters what lead to its fix
- Jonas Nyrup
  - reported a bug occuring when using `\cref` with comma separated arguments what lead to its fix
  - started an interesting discussion if cleveref's `\crefrange` command should be supported in autonum or not
  - found an incompatibility with biblatex, which lead to its fix
- Toby Cubitt
  - highlighted, that there are some users who might find valid use of cleveref's `\crefrange` command
- Marko Pinteric
  - found an underfull hbox error what lead to its removal
  - found the existance of a spurious whitespace problem what lead to its removal
- Heiko Oberdiek
  - found the reason of spurious whitespaces what lead to their removal
  - found an underfull hbox error what lead to its removal
- Carsten Grimm
  - found a bug when using `\ref` inside the `\caption` command which lead to its fix

- Ulrike Fischer
  - found that a `\protect` is missing to use `\ref` inside a `\caption`, which lead to its addition
- Johannes Gerer
  - found a bug when using `\cref` inside the `\section` command when hyperref is loaded which lead to its fix
- Michel Voßkuhle
  - found a bug when using autonum together with the subcaption package due to the missing support of starred reference commands in autonum which lead to its fix
  - sent an MWE showing a regression in the support of special characters what lead to its fix
- Pascal Germroth
  - sent an MWE showing a regression in the support of special characters what lead to its fix
- Vladimir Pozdyayev
  - found an error if an equation is used inside of a figure environment, which has not been fixed, yet
- Enno Nagel
  - highlighted poor areas in the documentation, which (hopefully) have been improved
- Deniz Stiegemann
  - found an incompatibility with biblatex, which lead to its fix
  - found a bug when having a newline directly followed by an alignment character in an align environment, which lead to its fix
- Fredrick Freekowtski
  - found a bug when referencing an align environment, which lead to its fix
  - found a bug when having a newline directly followed by an alignment character in an align environment, which lead to its fix
- TeX Stack Exchange user bers
  - found a bug when putting a label inside a split itself being inside of an equation, which lead to its fix

- Denis Bitouzé
  - found a bug when having a newline directly followed by an alignment character in an align environment, which lead to its fix

6 Implementation

The basic idea is to write into the aux file and save a variable whenever a label is referenced, so that the information is available in the current and in the next run. The label information is passed from the label command to the newline command. The newline command uses the label and the reference information to possibly add a `\notag`command, deciding if it is referenced or not.

`\csxdefaux` This command is similar to the `\csxdef` command from the package etoolbox, but instead of defining the command immediately, it is defined in the next run by writing it to the aux file. The name is given by the first argument (which may not have a leading backslash). The second argument is the replacement text. This command would be a candidate for inclusion into etoolbox. `\ifcsdef` can be used to check, if the command has been defined. See also http://tex.stackexchange.com/a/49035 Usage: `\csxdefaux{csname}{replacement}`

```
\def\csxdefaux#1#2{%
```

The expandafter commands are used to first expand the `\csname-\endcsname`. Then there is a command definition left, where the command and its replacement (which can also be a command) are protected by `\string` to create the command in the next run (when the aux file is read) and not in the current run (when the aux file is written).

```
\protected@write\@mainaux{}{%
```

The commented and the uncommented lines should do the same. The longer variant has the advantage, that there is no error the first run after the deactivation of the autonum package, as `\xdef`, read from the aux file, is always a known command, whereas `\csxdef` might not be known in that case, as with the deactivation of autonum it might happen, that etoolbox is not loaded anymore, too.

```
\csxdef{\detokenize{#1}}{#2}%
}%
}
```

Do not abort compilation, if the package has been deactivated from the last compilation to the current one and thus `\csxdef` might have become undefined (as etoolbox might not be loaded anymore). Therefore, add a dummy implementation of `\csxdef` to the aux file, which gets loaded if the real implementation is not available any longer.

```
\protected@write\@mainaux{}{%
\string\providecommand\string\csxdef[2]{}%
}
```

`\csxdefall` This command simply combines the commands `\csxdef` and `\csxdefaux`. This command would be a candidate for inclusion into etoolbox.

```
\def\csxdefall#1#2{%
\csxdefaux{#1}{#2}%
\csxdef{#1}{#2}%
}
```

`\CsLetLtxMacro` This command simply combines the functionality from `\cslet` and `\LetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd{\CsLetLtxMacro}[2]{%
\expandafter\LetLtxMacro\csname#1\endcsname#2%
}
```

`\LetCsLtxMacro` This command simply combines the functionality from `\letcs` and `\LetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd{\LetCsLtxMacro}[2]{%
\ifcsdef{#2}{%
\expandafter\LetLtxMacro\expandafter#1\csname#2\endcsname
}{%
\undef#1%
}%
}
```

`\CsLetCsLtxMacro` This command simply combines the functionality from `\csletcs` and `\LetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd*{\CsLetCsLtxMacro}[2]{%
\ifcsdef{#2}{%
\expandafter\LetLtxMacro\csname#1\expandafter\endcsname\csname#2\endcsname
}{%
\csundef{#1}%
}%
}
```

`\GlobalCsLetLtxMacro` This command simply combines the functionality from `\cslet` and `\GlobalLetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd{\GlobalCsLetLtxMacro}[2]{%
\expandafter\GlobalLetLtxMacro\csname#1\endcsname#2%
}
```

`\GlobalLetCsLtxMacro` This command simply combines the functionality from `\letcs` and `\GlobalLetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd{\GlobalLetCsLtxMacro}[2]{%
\ifcsdef{#2}{%
```

```
\expandafter\GlobalLetLtxMacro\expandafter#1\csname#2\endcsname
}{%
\undef#1%
}%
}
```

`\GlobalCsLetCsLtxMacro` This command simply combines the functionality from `\csletcs` and `\GlobalLetLtxMacro`. This command would be a candidate for inclusion into letltxmacro or possibly etoolbox.

```
\newrobustcmd*{\GlobalCsLetCsLtxMacro}[2]{%
\ifcsdef{#2}{%
\expandafter\GlobalLetLtxMacro\csname#1\expandafter\endcsname\csname#2\endcsname
}{%
\csundef{#1}%
}%
}
```

`\vanishprotect` This command encapsulates content to not appear in the table-of-contents or similar lists. The content is shown in normal text (captions or section headings and so on). The macro uses one arguments in spite of its parameterless definition.

```
\def\vanishprotect{%
\ifx\protect\@typeset@protect
\expandafter\@firstofone
\else
\expandafter\@gobble
\fi
}
```

`\ifcsedef` This command is similar to the `\ifcsdef` test from the package etoolbox, but the command sequence gets fully expanded before it is evaluated. This command would be a candidate for inclusion into etoolbox.

```
\def\ifcsedef#1#2#3{%
\edef\autonum@ifcsedefTemp{#1}%
\expandafter\ifcsdef\expandafter{\autonum@ifcsedefTemp}{#2}{#3}%
\undef{\autonum@ifcsedefTemp}%
}
```

`\newcommandsequence` This command combines the macro `\csdef` from the package etoolbox and LaTeX's `\newcommand` macro. This command would be a candidate for inclusion into etoolbox or into LaTeX2e, although the former is much more likely. The interface of `\newcommand` et al. is "moom" when using xparse syntax. Yet, `\newcommand` itself provides "ommm..." as interface, i.e. is not capable of defining itself. However, as all the argument types "m" and "o" are supported, combining multiple `\newcommand`s which read (curry) only part of the arguments are able to emulate the interface. Even simpler is the use of currying, as besides the first argument, everything is identical in `\newcommand` and `\newcommandsequence`. So the first arguments is read, and `\newcommand` is called to read the changed first argument and the rest.

```
\def\newcommandsequence#1{%
\expandafter\newcommand\csname #1\endcsname
}
% Tests:
% \newcommandsequence{testcommand}{This is a test}%
% \testcommand
% \newcommandsequence{testcommandtwo}[2]{This is a test with #1 and #2}%
% \testcommandtwo{one}{two}%
% \newcommandsequence{testcommandtwooptional}[2][three]{This is a test with #1 and #2}%
% \testcommandtwooptional[one]{two}%
% \testcommandtwooptional{two}%
```

`\renewcommandsequence` This command combines the macro `\csdef` from the package etoolbox and LaTeX's `\renewcommand` macro. This command would be a candidate for inclusion into etoolbox or into LaTeX2e, although the former is much more likely.

```
\def\renewcommandsequence#1{%
\expandafter\renewcommand\csname #1\endcsname
}
```

`\ifstar` This command is similar to the `\@ifstar` command from LaTeX, but it absorbs a token instead of peeking ahead. This command would be a candidate for inclusion into etoolbox.

```
% \def\ifstar#1#2#3{%
%  \ifstrequal{#1}{*}{#2}{#3}%
% }
```

`\csDeclareDocumentCommand` This command is a mixture of the `\csdef` command from the etoolbox package and the `\DeclareDocumentCommand` from the xparse package. This command would be a candidate for inclusion into xparse.

```
% \def\csDeclareDocumentCommand#1#2#3{%
%  \expandafter\DeclareDocumentCommand\csname #1\endcsname{#2}{#3}%
% }
```

`\autonum@debug` Activate and deactivate debugging by commenting and uncommenting the following code.

```
\def\autonum@debug#1{%
\PackageWarning{autonum}{#1}%
}
% \def\autonum@debug#1{}%
```

`\meaningx` Acts like `\meaning`, but if the given macro is defined via `\newcommand` or similar and has an optional argument, `\meaning` does always show something like `\@protected@testopt \command \\command`, which is not very useful. That is why `\meaningx` outputs the meaning of `\\command` instead. `\meaningx` is expandable (see "texdoc etex" and this information about expandability for more details).

```
\def\meaningx#1{%
\expandafter\ifcsname\string #1\endcsname
```

```
\expandafter\meaning\csname\string #1\endcsname
\else
\meaning#1%
\fi
}
```

\autonum@debug Print the \meaningx of the given macro.

```
\def\autonum@debugMacro#1{%
```

Please mind, that ppdflatex distorts the messages printed by this macro.

```
\autonum@debug{%
\detokenize{#1}defined as \MessageBreak
\meaning#1 \expandafter\ifcsname\string #1\endcsname\MessageBreak
with \expandafter\detokenize\expandafter{\csname\string#1\endcsname}defined as\MessageBreak
\meaningx#1\fi
}%
}
```

This is needed to not get overwritten by other packages. The package autonum only overwrites some commands whose name start with \autonum. Other commands are only patched, so the currently valid command gets called, too. So although not very polite, this behavior seems reasonable.

```
\AtBeginDocument{%
```

Most of amsmath's environments are redefined. The environments aligned and gathered are not redefined, as it is unclear, how the numbering should work.

```
%^^A\forcsvlist{\autonum@patchBlockEnvironment}{gathered,aligned}%
%^^A \ifdef{\multlined}{%
%^^A \autonum@patchBlockEnvironment{multlined}%
%^^A }{}%
```

If align is redefined before flalign or alignat, autonum.dtx does not build anymore. The reason of the error is unknown. As the error disappears when align is redefined after both, there is no motivation in finding the underlying problem.

```
\newlength{\autonum@environmentWidth}%
%  \forcsvlist{\autonum@patchParametrizedFullEnvironment}{alignat,figure}%
\forcsvlist{\autonum@patchParametrizedFullEnvironment}{alignat}%
\forcsvlist{\autonum@patchFullEnvironment}{equation,gather,multline,flalign,align}%
\autonum@patchBlockEnvironment{split}%
```

Patch the environment delimited by \[and \].

```
%  \autonum@patchShortcutEnvironment
\def\[#1\]{%
\begin{equation}#1\end{equation}%
}%
```

Support the normal \ref command and, if available, the \cref command from cleveref.

```
\autonum@generatePatchedReference{ref}%
\ifdef{\cref}{%
\autonum@generatePatchedReferenceCSL{cref}%
}{}%
```

```
}
```

`\autonum@patchEnvironment` Patch a mathematical environment to automatically show an equation's number, if a part is referenced and do not use a number otherwise. For completeness, the original definition (numbering every part of an displayed equation structure) is made available using a different name. Do not redefine the environments before getting the original label and newline commands. Use center as the default parameter, as a center environment is a neutral element regarding the subcommands' definitions.

```
% \def\autonum@patchEnvironment#1{
\def\autonum@patchFullEnvironment#1{%
%  \autonum@debug{patchFullEnvironment; 1=#1}%
\autonum@saveEnvironmentSubcommands{#1}{center}{}%
\autonum@patchEnvironmentHelper{#1}{0}%
}
\def\autonum@patchParametrizedFullEnvironment#1{%
%  \autonum@debug{patchParametrizedFullEnvironment; 1=#1}%
\autonum@saveEnvironmentSubcommands{#1}{center}{1}%
\autonum@patchEnvironmentHelper{#1}{1}%
}
\def\autonum@patchBlockEnvironment#1{%
%  \autonum@debug{patchBlockEnvironment; 1=#1}%
```

There is no need to patch the newline or label commands like for the other environments, as they already have been patched in those outer math environments. So the environment subcommands do not have to be saved, too. However, split inside of an equation environment resets the label command, so that our version of the label command (of the equation) is no longer valid. Currently, this is taken care of in startChangeEnvironment.

```
\autonum@renameEnvironment{#1}{0}%
\autonum@changeEnvironment{#1}{0}%
}
\def\autonum@patchEnvironmentHelper#1#2{%
%  \autonum@debug{patchEnvironmentHelper; 1=#1; 2=#2}%
\autonum@renameEnvironment{#1}{#2}%
\autonum@changeEnvironment{#1}{#2}%
\autonum@generatePatchedLabel{#1}%
\autonum@generatePatchedNewline{#1}%
}
```

`onum@saveEnvironmentSubcommands` This macro saves the newline code used in a mathematic display environment so that it can be used later. This is necessary, as saving it in the instance of the environment, where it should be used, does not work. Allow for two arguments, to enable putting the alignment building blocks into an equation environment (see amsmath documentation). As a neutral element, the center environment can be used. The third argument is needed for environments which have arguments themself.

```
\def\autonum@saveEnvironmentSubcommands#1#2#3{%
```

```
%  \autonum@debug{In saveEnvironmentSubcommands; 1=#1; 2=#2; 3=#3}%
\begin{textblock}{1}[1,1](0,0)%
\begin{#2}%
\begin{#1}#3%
```

Avoid underfull hbox warning in multline, by putting content of the correct size in it. The correct size can only be measured here, as the values might change due to the beginning of environments.

```
\deflength{\autonum@environmentWidth}{\linewidth-\multlinegap-\multlinegap}%
\hspace{\autonum@environmentWidth}%
```

Using global here is necessary to get the information out of the environment. As `\\` has an optional argument, you would think, that `\GlobalCsLetLtxMacro` is correct, while `\global\cslet` is incorrect (see here). But although `\\` has an optional argument, it does not use `\newcommand` (see the definintion of `\Let@` in amsmath.sty for the details), but plain `\TeX`. Although using LetLtxMacro should never no harm, it results in errors here, so it isn't used.

```
\global\cslet{autonum@newline#1}\\%
```

Use `\notag` to not increase the equation counter (otherwise the first equation shown would not have number 1).

```
\notag
```

For multline, check, that autonum's label command is undefined, because this means, that the first pass (measuring pass) of the environment is active. This is to avoid getting the `\label` command of the second pass (displaying pass), where the `\label` command is set to the null definition. The multline environment seems to need the first pass, whereas other environments seem to need the second pass, so adapt to the environments. The following code does the macro definition twice for the other environments - once for the measurement pass and once for the display pass. This is not a problem, as the result of the first pass is overwritten with the correct definition in the second pass.

```
\ifboolexpr{not test {\ifstrequal{#1}{multline}} or test {\ifcsundef{autonum@label#1}}}{%
%  \autonum@debug{In saveEnvironmentSubcommands, define autonum@label#1 to \meaning\label}%
\GlobalCsLetLtxMacro{autonum@label#1}\label
}{}%
\end{#1}%
\end{#2}%
\end{textblock}%
}
```

`\autonum@renameEnvironment` Rename the old environment to be accessible with an appended + by saving the original environment using a different name. The first argument contains the environment's name, the second argument contains the number of arguments the environment has.

```
\def\autonum@renameEnvironment#1#2{%
\csletcs{autonum@#1Old}{#1}%
\csletcs{autonum@end#1Old}{end#1}%
\newenvironment{#1+}[#2]{%
```

```
\csuse{autonum@#1Old}%
}{%
\csuse{autonum@end#1Old}%
}%
}
```

`\autonum@changeEnvironment` Now change the environment. This command only supports displayed equation structures and is not suited for other environments (as e.g. figures). The second argument contains the number of arguments the redefined environment has.

```
\def\autonum@changeEnvironment#1#2{%
%  \autonum@debug{changeEnvironment; 1=#1; 2=#2}%
```

Although Amsmath's environment is executed twice (for measuring and for painting), the content here is executed only once. The following if is only needed to distinguish between environments without (e.g. equation) and with one parameter (e.g. alignat).

```
\ifnum #2=0%
\renewenvironment{#1}{%
\autonum@startChangeEnvironment{#1}{}%
}{%
\autonum@endChangeEnvironment{#1}%
}%
\else
\renewenvironment{#1}[1]{%
\autonum@startChangeEnvironment{#1}{##1}%
}{%
\autonum@endChangeEnvironment{#1}%
}%
\fi
```

Delete the starred versions of the environment, as they sometimes lead to strange errors a long time after using the starred version. By deleting it, the error occurs at the right place.

```
\global\csundef{#1*}%
\global\csundef{end#1*}%
}
```

`\autonum@startChangeEnvironment` Start the changed environment.

```
\def\autonum@startChangeEnvironment#1#2{%
%  \autonum@debug{startChangeEnvironment; 1=#1; 2=#2}%
```

Prepare the label and the newline commands and begin the displayed equation environment.

```
\ifstrequal{#1}{split}{%
```

Split does not have its own label command. However, beginning the split environment inside of an equation environment resets the label command so that our definition is no longer used. Thus, save the (currently patched) label command before starting the environment.

```
\GlobalLetLtxMacro\autonum@outerMathEnvironmentLabel\label
```

```
}{%
\autonum@saveSubcommands
}%
%
\csuse{autonum@#1Old}#2%
%
\ifstrequal{#1}{split}{%
```

Restore the just saved label command (see above).

```
\GlobalLetLtxMacro\label\autonum@outerMathEnvironmentLabel
}{%
\autonum@patchSubcommands{#1}%
}%
}
```

`\autonum@endChangeEnvironment` Close the changed environment.

```
\def\autonum@endChangeEnvironment#1{%
%  \autonum@debug{In endChangeEnvironment; 1=#1}%
```

Possibly hide the number of the last equation in the displayed equation environment and end the latter one.

```
\autonum@possiblyHideNumber
\csuse{autonum@end#1Old}%
```

Restore the subcommands. Do not restore the subcommands for the split environment, as `\utonum@startChangeEnvironment` is not called (even if set) for the split environment due to yet unknown reasons. The hypothesis is, that this is as the split environment is inside of another environment (equation). Thus, `\renewenvironment` is called inside a group. As `\renewenvironment` is always local (see http://tex.stackexchange.com/q/51733), the redefinition of the split environment does not work. As the split subcommands are currently neither saved nor patched, it is not only unnecessary, but really harmful to restore the subcommands. Doing so breaks splits together with other material in an align environment. This code should be changed after the root problem has been fixed. Neither delete the label, as it might be defined before the split environment and is used after its end.

```
\ifstrequal{#1}{split}{%
```

If a split environment is inside an align environment, the `\notag` must be set inside of the split environment to get correct vertical spacing after the environment if the line is so long, that the equation number would be placed below the equation and if the equation is not referenced. In order to set the `\notag` it must be known if the equation is referenced or not. With the current logic, the label is used to identify an equation. If the label is only given after the split (but before the line of the align ends), this is impossible, thus, the label has to be given before the end of the split environment. As split provides no numbering, it's best to provide the label before the beginning of the split environment. To save users, redefine `\label` to result in an error if the `\label` is called too late. In the future, this could be changed if equations have another unique identifier like a number. The `\label`

can be overwritten (but mind the optional argument, which must be supported), as the split TeX-group will end directly after this macro. In order to be effective, it must be smuggled outside, which `\aftergroup@def` from etextools takes care of. Outside, it will be removed automatically, as soon as the TeX-group ends, e.g. if the line or the environment ends.

```
\renewcommand\label[2][]{%
\PackageError{autonum}{%
A label must not be placed between \detokenize{\end{split}}\MessageBreak
and the end (of a line) of a math environment\MessageBreak
when using autonum. Wrongly placed label is:\MessageBreak
##2%
}{Move the \detokenize{\label}call before \detokenize{\begin{split}}}%
}%
\aftergroup@def{\label}%
}{%
\autonum@restoreSubcommands
\global\undef\autonum@currentLabel
}%
%  \autonum@debug{In endChangeEnvironment at the end}%
}
```

`\autonum@saveSubcommands` Save the current newline and label commands.

```
\def\autonum@saveSubcommands{%
\GlobalLetLtxMacro\autonum@labelNormal\label
\GlobalLetLtxMacro\autonum@newlineNormal\\%
}
```

`\autonum@patchSubcommands` Patch the label command, as some special data has to be saved with each usage. In order to support multi-line equations, the counter must be increased in every line, as every line is a possible reference target. Therefore,
has to be overwritten, too. This must be global, as amsmath is very annoying with overwriting local definitions, e.g. in align environments.

```
\def\autonum@patchSubcommands#1{%
```

Activate the patched label command and support an optional argument for the `\label` command.

```
\GlobalLetCsLtxMacro\label{autonum@patched#1Label}%
```

Do not patch the newline command in a multline environment, as only the last line may get a `\notag` command, because all lines basically build one equation (see also amsmath's documentation, section 3.3).

```
\ifstrequal{#1}{multline}{%
}{%
\GlobalLetCsLtxMacro\\{autonum@patched#1Newline}%
}%
}
```

`\autonum@restoreSubcommands` Restore the newline and label commands. This must be global, as it had been overwritten globally in `\autonum@patchSubcommands`.

```
\def\autonum@restoreSubcommands{%
%  \autonum@debug{In restoreSubcommands}%
\GlobalLetLtxMacro\label\autonum@labelNormal
\GlobalLetLtxMacro\\\autonum@newlineNormal
%  \autonum@debug{In restoreSubcommands at the end}%
}
```

`\autonum@generatePatchedLabel` Use an extra command to patch the used label command for efficiency.

```
\def\autonum@generatePatchedLabel#1{%
```

As the `\label` command can have an optional argument (see cleveref: Overriding the Cross-Reference Type), it must be supported in the patched version of the command, too. As the basic version of `\label` does not support an optional argument (see texdoc source2e), inside of the following macro the label command must not be called with a possible empty optional argument unconditionally. Instead, there must be a test if an optional argument has been provided and only then it must be used in the call of the original label command. Use the trick with the empty macro as a test.

```
\newcommandsequence{autonum@patched#1Label}[2][]{%
%  \autonum@debug{In patched#1Label, 1=##1, 2=##2}%
```

Do not do the labeling work twice, i.e. in amsmath's measuring and in it's display pass. Doing everything in the display pass is correct, because during the measuring pass, the original label commands are set to only gobble their arguments instead of doing something useful (the latter is done in the display pass).

```
\ifmeasuring@
\else
```

The labeling information is needed in the newline command. Therefore, the following variable is used to store it until the next newline command. As the definition is local and every line in an multi-line displayed math environment has its own group, the variable does not have to be deleted explicitly.

```
\ifdef{\autonum@currentLabel}{%
```

As `\autonum@currentLabel` can be a global macro, there was a false positive error with math environments which are executed twice. To avoid this false positives, only check during the first run of these environments.

```
\ifmeasuring@
\PackageError{autonum}{Two succeeding \string\label's detected}{Did you forget a \string\\?}%
\fi
}{%
```

Define the label globally, as otherwise it gets deleted inside of an align environment if it is defined before a split environment and should be applied after the split environment. Thus, it is globally deleted when ending a math environment or a line in such an environment (always with the exception of the split environment).

```
\ifboolexpr{test {\ifstrequal{#1}{flalign}} or test {\ifstrequal{#1}{alignat}}}{%
```

Do not define the label globally for flalign and alignat environment, as it results in warnings. According to the amsmath documentation it's uncommon to use split

inside of both environments, so this should be an accectable compromise. Deleting globally when only defining locally does not seem to be a problem.

```
\def\autonum@currentLabel{##2}%
}{%
%  \autonum@debug{In patched#1Label, currentLabel globally defined to ##2}%
\global\def\autonum@currentLabel{##2}%
}%
}%
```

Only call the original label command if the label gets referenced. This obviously is identical if the reference is located before the label. It is also identical if the reference is located after the label, as the `\\` or `\endenvironment` commands which follow the `\label` would suppress the numbering anyway in the first pass. In the second pass, the information about referencing is the same as if only the content of the following if-command were available.

```
\ifcsedef{autonum@##2Referenced}{%
%  \autonum@debug{In patched#1Label, label ##2 has been referenced}%
```

The environment's original label command is called to do the real labeling. As it checks for erroneous succeeding labels using `\f@label`, this variable has to be emptied before every call.

```
\let\df@label\@empty%
```

If the following test is true, `\label` has been called without optional argument, as the default has been used internally. This test is from egreg.

```
%  \autonum@debug{In patched#1Label, autonum@label#1 is called with mandatory argument ##2}%
\if\relax\detokenize{##1}\relax
%  \autonum@debug{In patched#1Label, no optional argument given}%
\csuse{autonum@label#1}{##2}%
\else
\csuse{autonum@label#1}[##1]{##2}%
\fi
}{%
%  \autonum@debug{In patched#1Label, label ##2 has not been referenced}%
}%
\fi
}%
}
```

`\autonum@generatePatchedNewline` This command generates patched newline commands for displayed math environments, so that they can simply be activated when needed.

```
\def\autonum@generatePatchedNewline#1{%
```

The `\\` command has an optional argument, e.g. in an align environment, to put some extra vertical space between two lines. Thus, the patched newline command must correctly cope with an optional argument, too, if it has been given.

```
\newcommandsequence{autonum@patched#1Newline}{%
%  \autonum@debug{In patched#1Newline, 1=##1}%
\autonum@possiblyHideNumber
\global\undef\autonum@currentLabel
```

```
%  \autonum@debug{\expandafter\meaning\csname autonum@newline#1\endcsname}%
\csuse{autonum@newline#1}%
```

CAUTION: Do not write code below this comment and inside of this macro. This is because the newline command above looks for an optional argument in the input stream, which it cannot read if there are further tokens inside of this macro. Absorbing the optional argument in this macro and transfering it to the newline command did have unexpected side effects (a test case has been added for this).

```
}%
}
```

`\autonum@possiblyHideNumber` Define this command, which can hide the current line's number if the label is not referenced.

```
\def\autonum@possiblyHideNumber{
\ifdef{\autonum@currentLabel}{%
%  \autonum@debug{In possiblyHideNumber, existing currentLabel=\autonum@currentLabel}%
\ifcsedef{autonum@\autonum@currentLabel Referenced}{%
}{%
\notag
}%
```

The current label does not have to be cleaned, as every line is a separate cell defining a local group in an displayed math environment.

```
}{%
\notag
}%
}
```

`utonum@generatePatchedReference` This command can patch reference commands with a normal input argument.

```
\def\autonum@generatePatchedReference#1{%
\autonum@generatePatchedReferenceGeneral{#1}{autonum@use}%
}
```

`num@generatePatchedReferenceCSL` This command can patch reference commands which expect a comma separated list as input argument.

```
\def\autonum@generatePatchedReferenceCSL#1{%
\autonum@generatePatchedReferenceGeneral{#1}{forcsvlist}%
}
```

`\skipInPDFTOC` The content of the macro is never written to the PDF's table of content (TOC). This is especially useful for macros which should only modify the normal PDF's content.

```
\def\skipInPDFTOC#1{%
```

If hyperref is loaded (i.e. `\texorpdfstring` is defined), take care to only write into the document. If it is not loaded, this is the default behavior.

```
\ifdef{\texorpdfstring}{%
\texorpdfstring{#1}{}%
}{%
```

```
#1%
}%
}
```

generatePatchedReferenceGeneral

This command can patch arbitrary reference commands. The patch logic can be different to patching the label command, as the references have to be patched only once, so optimizing for speed is counter-productive.

```
\def\autonum@generatePatchedReferenceGeneral#1#2{%
```

There might be reference commands, which have an optional argument. To support these, use `\CsLetCsLtxMacro` instead of `\csletcs`.

```
\CsLetCsLtxMacro{autonum@reference#1Old}{#1}%
```

Do not absorb any arguments, yet, because every further processing must be protected and this is the simplest way to achieve protected processing. Something like using `\@ifstar` inside of a moving argument would otherwise be very hard or impossible. Unfortunately, a simple approach like `\csDeclareDocumentCommand{#1}{sm}` does produce a not yet understood infinite recursion.

```
\csdef{#1}{%
```

Use protect, to avoid problems with sections and captions. This is the same trick that hyperref uses, which can be seen by executing `\show\ref`.

```
\protect\autonum@processReference{autonum@reference#1Old}{#2}%
}%
}
```

\autonum@processReference

If the reference command supports a starred call, as `\ref` when loading hyperref does for example, the star must be supported, as well. This macro separates calls using a star from those not using one.

```
\def\autonum@processReference#1#2{%
\@ifstar{%
\autonum@processReferenceHelper{#1}{#2}{*}%
}{%
\autonum@processReferenceHelper{#1}{#2}{}%
}%
}%
```

Do not abort compilation, if the package has been deactivated from the last compilation to the current one and thus the table of content or the list of figure files might reference the now undefined `\autonum@processReference` command. Therefore, add a dummy implementation to both files, which gets loaded if the real implementation is not available any longer. The dummy implementation is only written, if the respective files exist (are otherwise used).

```
\addtocontents{toc}{%
\string\providecommand\string\autonum@processReference[2]{}%
}
\addtocontents{lof}{%
\string\providecommand\string\autonum@processReference[2]{}%
}
```

\autonum@processReference Mark label as referenced and call the old reference command. Do not write in the PDF's TOC, as this would lead to a write inside of a write when, e.g., calling `\cref` inside of a section command. Also use `\vanishprotect` to avoid writing `\autonum@markLabelAsReferenced` into the aux file or similar files. This avoids errors after deactivating the autonum package.

```
% \autonum@processReferenceHelper{reference command name}{extraction command name}{star or emp
\def\autonum@processReferenceHelper#1#2#3#4{%
%  \autonum@debug{In processReferenceHelper, 1=#1, 2=\detokenize{#2}, 3=#3, 4=#4}%
\skipInPDFTOC{%
\vanishprotect{%
\csuse{#2}{\protect\autonum@markLabelAsReferenced}{#4}%
}%
}%
\csuse{#1}#3{#4}%
}%
```

\autonum@markLabelAsReferenced This is a simple helper macro to mark a label as referenced. The reference information is stored into a variable (for the current run) and into the aux file (for the next run), so it does not matter if the reference is used before or after the definition of the label. Saving into a variable saves one compilation run, although still up to three are needed to get everything right.

```
\def\autonum@markLabelAsReferenced#1{%
%  \autonum@debug{In markLabelAsReferenced, autonum@#1Referenced defined}%
\csxdefall{autonum@#1Referenced}{}%
}
```

\autonum@use This is a simple helper macro which can be used like a function handle similar to csuse but expecting a macro instead of a macro's name.

```
\def\autonum@use#1#2{%
#1{#2}%
}
```

utonum@patchShortcutEnvironment Use a counter to numerate all `\[-\]`-environments linearly.

```
\newcounter{autonum@counter}
```

`\[` and `\]` are redefined as the correct one of `equation` and `align`. Due to the improved numbering, the old environment's capabilities are basically a subset of the new capabilities.

```
\def\autonum@patchShortcutEnvironment{%
\def\[##1\]{%
```

This command checks if the current environment only consists of one line without counting lines in sub-environments. The default will result in an align environment, as incorrectly using an equation instead of an correct align results in a compile error.

```
\ifcsedef{autonum@\Roman{autonum@counter}HasExactlyOneLine}{%
\autonum@useWithMultipleLineDetection{equation}{##1}%
}{%
\autonum@useWithMultipleLineDetection{align}{##1}%
```

```
}%
\stepcounter{autonum@counter}%
}%
}
```

`um@useWithMultipleLineDetection` This function uses an environment defined by the first argument to display the content given in the second argument. A multiple-line detection is activated, to set a variable if more than one line is used.

```
\def\autonum@useWithMultipleLineDetection#1#2{%
\begin{#1}%
\autonum@patchParentheses
```

Use global as this is in the middle of the first local group of a math environment.

```
\GlobalLetLtxMacro\autonum@patchedNewline\\%
```

Set the multipleLines variable if a newline is used. Do not use the newline for equations, as this results in "There's no line here to end" errors. This is ok, as if there is a newline in an environment, which is currently an equation, it is wrong anyway and should be set as an align environment. For that it is enough to set the multipleLines variable.

```
\ifstrequal{#1}{align}{%
\gdef\\{%
\autonum@patchedNewline
\gdef\autonum@multipleLines{}%
}%
}{%
\gdef\\{%
\gdef\autonum@multipleLines{}%
}%
}
```

Set the environment's content and reset the newline command.

```
#2%
\GlobalLetLtxMacro\\\autonum@patchedNewline
```

Store information if the current math environment. The roman number is used, as there might be no label and if there is one, it might not be available at the beginning of the environment. Delete the multipleLines variable, to avoid influencing the next `\[-\]`-environment, as the variable must be global.

```
\ifdef{\autonum@multipleLines}{%
\global\undef{\autonum@multipleLines}%
}{%
\csxdefaux{autonum@\Roman{autonum@counter}HasExactlyOneLine}{\Roman{autonum@counter}}%
}%
\autonum@restoreParentheses
\end{#1}%
}
```

`\autonum@patchParentheses` This function patches the left and the right parentheses.

```
\global\def\autonum@patchParentheses{%
```

```
\autonum@patchParenthesis{(}{Left}{}%
\autonum@patchParenthesis{)}{Right}{end}%
}
```

`\autonum@patchParenthesis` This function patches a parenthesis given in the first argument with a name partly given in the second argument by using the third argument.

```
\global\def\autonum@patchParenthesis#1#2#3{%
\ifcsdef{#1}{%
\global\csletcs{autonum@old#2Parenthesis}{#1}%
}{}%
\global\csletcs{#1}{#3split}%
}
```

`\autonum@restoreParentheses` This function restores the left and the right parentheses.

```
\global\def\autonum@restoreParentheses{%
\autonum@restoreParenthesis{(}{Left}%
\autonum@restoreParenthesis{)}{Right}%
}
```

`\autonum@restoreParenthesis` This function restores a parenthesis given in the first argument with the name given in the second argument.

```
\global\def\autonum@restoreParenthesis#1#2{%
\ifcsdef{autonum@old#2Parenthesis}{%
\global\csletcs{#1}{autonum@old#2Parenthesis}%
\global\csundef{autonum@old#2Parenthesis}%
}{}%
}
```