The package nicematrix

F. Pantigny
fpantigny@wanadoo.fr
October 20, 2020

Abstract

The \LaTeX\ package nicematrix provides new environments similar to the classical environments \{tabular\}, \{array\} and \{matrix\} of array and amsmath but with extended features.

\begin{center}
\begin{tabular}{cll}
\hline
Product & dimensions (cm) & Price \\
\hline
small & 3 & 5.5 & 1 & 30 \\
standard & 5.5 & 8 & 1.5 & 50.5 \\
premium & 8.5 & 10.5 & 2 & 80 \\
extra & 8.5 & 10 & 1.5 & 85.5 \\
special & 12 & 12 & 0.5 & 70 \\
\hline
\end{tabular}
\end{center}

The package nicematrix is entirely contained in the file nicematrix.sty. This file may be put in the current directory or in a texmf tree. However, the best is to install nicematrix with a \TeX\ distribution as MiKTeX or TeXlive.

This package can be used with xelatex, lualatex, pdflatex but also by the classical workflow latex-dvips-ps2pdf (or Adobe Distiller).

This package requires and loads the packages l3keys2e, xparse, array, amsmath, pgfcore and the module shapes of PGF (tikz, which is a layer over PGF is not loaded). The final user only has to load the package with \texttt{\usepackage{nicematrix}}.

The idea of nicematrix is to create PGF nodes under the cells and the positions of the rules of the tabular created by array and to use these nodes to develop new features. As usual with PGF, the coordinates of these nodes are written in the .aux to be used on the next compilation and that’s why nicematrix may need several compilations.

Most features of nicematrix may be used without explicit use of PGF or Tikz (which, in fact, is not loaded by default).

A command \texttt{\NiceMatrixOptions} is provided to fix the options (the scope of the options fixed by this command is the current \TeX\ group: they are semi-global).

Important

Since the version 5.0 of nicematrix, one must use the letters l, c and r in the preambles of the environments and no longer the letters L, C and R.

For sake of compatibility with the previous versions, there exists an option define-L-C-R which must be used when loading nicematrix.

\texttt{\usepackage[define-L-C-R]{nicematrix}}

\footnote{This document corresponds to the version 5.5 of nicematrix, at the date of 2020/10/20.}
1 The environments of this package

The package nicematrix defines the following new environments.

\{NiceTabular\}  \{NiceArray\}  \{NiceMatrix\}
\{NiceTabular*\}  \{pNiceArray\}  \{pNiceMatrix\}
\{bNiceArray\}  \{bNiceMatrix\}
\{BNiceArray\}  \{BNiceMatrix\}
\{vNiceArray\}  \{vNiceMatrix\}
\{VNiceArray\}  \{VNiceMatrix\}

The environments \{NiceArray\}, \{NiceTabular\} and \{NiceTabular*\} are similar to the environments \{array\}, \{tabular\} and \{tabular*\} of the package \texttt{array} (which is loaded by \texttt{nicematrix}).

The environments \{pNiceArray\}, \{bNiceArray\}, etc. have no equivalent in \texttt{array}.

The environments \{NiceMatrix\}, \{pNiceMatrix\}, etc. are similar to the corresponding environments of \texttt{amsmath} (which is loaded by \texttt{nicematrix}): \{matrix\}, \{pmatrix\}, etc.

All the environments of the package \texttt{nicematrix} accept, between square brackets, an optional list of key=value pairs. \textbf{There must be no space before the opening bracket (\{} of this list of options.}

\textbf{Important}

Before the version 5.0, it was mandatory to use, for technical reasons, the letters \texttt{L}, \texttt{C} et \texttt{R} instead of \texttt{l}, \texttt{c} et \texttt{r} in the preambles of the environments of \texttt{nicematrix}. If we want to be able to go on using these letters, \texttt{nicematrix} must be loaded with the option \texttt{define-L-C-R}.

\usepackage[{define-L-C-R}]{nicematrix}

2 The vertical space between the rows

It's well known that some rows of the arrays created by default with \LaTeX are, by default, too close to each other. Here is a classical example.

\begin{pmatrix}
\frac12 & -\frac12 \\ 
\frac13 & \frac14 \\
\end{pmatrix}

Inspired by the package \texttt{cellspace} which deals with that problem, the package \texttt{nicematrix} provides two keys \texttt{cell-space-top-limit} and \texttt{cell-space-bottom-limit} similar to the parameters \texttt{cellspacetoplimit} and \texttt{cellspacebottomlimit} of \texttt{cellspace}. The initial value of these parameters is 0 pt in order to have for the environments of \texttt{nicematrix} the same behaviour as those of \texttt{array} and \texttt{amsmath}. However, a value of 1 pt would probably be a good choice and we suggest to set them with \texttt{NiceMatrixOptions}.\footnote{One should remark that these parameters apply also to the columns of type \texttt{S} of \texttt{siunitx} whereas the package \texttt{cellspace} is not able to act on such columns of type \texttt{S}.}

\begin{NiceMatrixOptions}{cell-space-top-limit = 1pt,cell-space-bottom-limit = 1pt}
\begin{pmatrix}
\frac12 & -\frac12 \\ 
\frac13 & \frac14 \\
\end{pmatrix}
\end{NiceMatrixOptions}

\begin{NiceMatrix}
\frac12 & -\frac12 \\ 
\frac13 & \frac14 \\
\end{pmatrix}
\end{NiceMatrix}
3 The vertical position of the arrays

The package `nicematrix` provides an option `baseline` for the vertical position of the arrays. This option takes in as value an integer which is the number of the row on which the array will be aligned.

\[
A = \begin{pNiceMatrix}[baseline=2]
\frac{1}{\sqrt{1+p^2}} & p & 1-p \\
1 & 1 & 1 \\
1 & p & 1+p 
\end{pNiceMatrix}
\]

It’s also possible to use the option `baseline` with one of the special values `t`, `c` or `b`. These letters may also be used absolutely like the option of the environments `{tabular}` and `{array}` of `array`. The initial value of `baseline` is `c`.

In the following example, we use the option `t` (equivalent to `baseline=t`) immediately after an `\item` of list. One should remark that the presence of a `\hline` at the beginning of the array doesn’t prevent the alignment of the baseline with the baseline of the first row (with `{tabular}` or `{array}` of `array`, one must use `\firsthline`.

\begin{enumerate}
\item an item
\smallskip
\item \renewcommand{\arraystretch}{1.2}
\begin{NiceArray}[t]{lcccccc}
\hline
n & 0 & 1 & 2 & 3 & 4 & 5 \\
u_n & 1 & 2 & 4 & 8 & 16 & 32 \\
\hline
\end{NiceArray}
\end{enumerate}

However, it’s also possible to use the tools of `booktabs`: `\toprule`, `\bottomrule`, `\midrule`, etc.

\begin{enumerate}
\item an item
\smallskip
\item \begin{NiceArray}[t]{lcccccc}
\toprule
n & 0 & 1 & 2 & 3 & 4 & 5 \\
u_n & 1 & 2 & 4 & 8 & 16 & 32 \\
\bottomrule
\end{NiceArray}
\end{enumerate}

It’s also possible to use the key `baseline` to align a matrix on a horizontal rule (drawn by `\hline`). In this aim, one should give the value `line-i` where `i` is the number of the row following the horizontal rule.

\NiceMatrixOptions{cell-space-top-limit=1pt,cell-space-bottom-limit=1pt}

\[
A = \begin{pNiceMatrix}[baseline=line-3]
\frac{1}{A} & \frac{1}{B} & 0 & 0 \\
1 & 1 & 0 & 0 \\
\frac{C}{D} & \frac{D}{D} & 0 & 0 \\
0 & 0 & A & B \\
0 & 0 & D & D 
\end{pNiceMatrix}
\]
4 The blocks

4.1 General case

In the environments of nicematrix, it’s possible to use the command \Block in order to place an element in the center of a rectangle of merged cells of the array.

The command \Block must be used in the upper leftmost cell of the array with two arguments. The first argument is the size of the block with the syntax $i-j$ where $i$ is the number of rows of the block and $j$ its number of columns. The second argument is the content of the block.

In \{NiceTabular\} the content of the block is composed in text mode. In the other environments, it is composed in math mode.

Here is an example of utilisation of the command \Block in mathematical matrices.

$$\begin{bNiceArray}{ccc|c}[margin] \Block{3-3}{A} & & & 0 \\ & \hspace{1cm} & & \Vdots \\ & & & 0 \\ \hline 0 & \Cdots & 0 & 0 \end{bNiceArray}$$

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it’s not possible to use directly a command like \Large, \Large and \LARGE. That’s why the command \Block provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.

$$\begin{bNiceArray}{ccc|c}[margin] \Block{3-3}<>{\Large}{A} & & & 0 \\ & \hspace{1cm} & & \Vdots \\ & & & 0 \\ \hline 0 & \Cdots & 0 & 0 \end{bNiceArray}$$

New 5.3 It’s possible to set the horizontal position of the block with one of the keys l, c and r.

$$\begin{bNiceArray}{ccc|c}[margin] \Block[r]{3-3}<>{\LARGE}{A} & & & 0 \\ & \hspace{1cm} & & \Vdots \\ & & & 0 \\ \hline 0 & \Cdots & 0 & 0 \end{bNiceArray}$$

One must remark that, by default, the commands \Blocks don’t create space (excepted, to some extent, the mono-column blocks: see below).

In the following example, we have had to enlarge by hand the columns 2 and 3 (with the construction \textwidth{...} of array).

\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose & tulipe & marguerite & dahlia \\
violette & \Block{2-2}<>{\LARGE}\color{blue}De très jolies fleurs & & souci \\
pervenche & & lys \\
arum & iris & jacinthe & muguet
\end{NiceTabular}
4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.
- The specification of the horizontal position provided by the type of column (c, r or l) is taken into account for the blocks.
- The specifications of font specified for the column by a construction \{\ldots\} in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

\begin{NiceTabular}{@{}>{\bfseries}lr@{}} 
\hline
\Block{2-1}{John} & 12 \\
& 13 \hline
Steph & 8 \\
\Block{3-1}{Sarah} & 18 \\
& 17 \\
& 15 \hline
Ashley & 20 \\
Henry & 14 \\
\Block{2-1}{Madison} & 15 \\
& 19 \hline
\end{NiceTabular}

4.3 A small remark

One should remark that the horizontal centering of the contents of the blocks is correct even when an instruction such as \{\qquad\} has been used in the preamble of the array in order to increase the space between two columns (this is not the case with \multicolumn). In the following example, the header “First group” is correctly centered.

\begin{NiceTabular}{@{}c!\{\qquad\}ccc!\{\qquad\}ccc@{}} 
\toprule
& \Block{1-3}{First group} & & & \Block{1-3}{Second group} \\
Rank & 1A & 1B & 1C & 2A & 2B & 2C \\
\midrule
1 & 0.657 & 0.913 & 0.733 & 0.830 & 0.387 & 0.893 \\
2 & 0.343 & 0.537 & 0.655 & 0.690 & 0.471 & 0.333 \\
3 & 0.783 & 0.885 & 0.015 & 0.306 & 0.643 & 0.263 \\
4 & 0.161 & 0.708 & 0.386 & 0.257 & 0.074 & 0.336 \\
\bottomrule
\end{NiceTabular}
5 The rules

The usual techniques for the rules may be used in the environments of \texttt{nicematrix} (excepted \texttt{\vline}). However, there is some small differences with the classical environments.

5.1 Some differences with the classical environments

5.1.1 The vertical rules

In the environments of \texttt{nicematrix}, the vertical rules specified by \texttt{|} in the preambles of the environments are never broken, even by an incomplete row or by a double horizontal rule specified by \texttt{\hline} (there is no need to use \texttt{\textbackslash hhline}).

\begin{NiceTabular}{|c|c|} \hline
First & Second \\
\hline
Peter \\ \hline
Mary & George\\ \hline
\end{NiceTabular}

However, the vertical rules are not drawn in the blocks.

If you use \texttt{booktabs} (which provides \texttt{\textbackslash toprule}, \texttt{\textbackslash midrule}, \texttt{\textbackslash bottomrule}, etc.) and if you really want to add vertical rules (which is not in the spirit of \texttt{booktabs}), you should notice that the vertical rules drawn by \texttt{nicematrix} are compatible with \texttt{booktabs}.

\begin{NiceArray}{|cccc|} \toprule
a & b & c & d \\
\midrule
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\bottomrule
\end{NiceArray}

However, it’s still possible to define a specifier (named, for instance, \texttt{I}) to draw vertical rules with the standard behaviour of \texttt{array}.

\begin{tabular}{cccc}
A&B&C&D \\
\cline{2-2}
A&B&C&D \\
\end{tabular}

This is the behaviour since the version 5.1 of \texttt{nicematrix}. Prior to that version, the behaviour was the standard behaviour of \texttt{array}.

5.1.2 The command \texttt{\textbackslash cline}

The horizontal and vertical rules drawn by \texttt{\hline} and the specifier “|” make the array larger or wider by a quantity equal to the width of the rule (with \texttt{array} and also with \texttt{nicematrix}).

For historical reasons, this is not the case with the command \texttt{\textbackslash cline}, as shown by the following example.

\begin{tabular}{cccc}
A&B&C&D \\
\cline{2-2}
A&B&C&D \\
\end{tabular}

In the environments of \texttt{nicematrix}, this situation is corrected (it’s still possible to go to the standard behaviour of \texttt{\textbackslash cline} with the key \texttt{standard-\textbackslash cline}).

\begin{tabular}{cccc}
A&B&C&D \\
\cline{2-2}
A&B&C&D \\
\end{tabular}

\footnote{This is the behaviour since the version 5.1 of \texttt{nicematrix}. Prior to that version, the behaviour was the standard behaviour of \texttt{array}.}
5.2 The thickness and the color of the rules

The environments of \texttt{nicematrix} provide a key \texttt{rules/width} to set the width (in fact the thickness) of the rules in the current environment. In fact, this key merely sets the value of the length \texttt{\arrayrulewidth}.

It’s well known that \texttt{colortbl} provides the command \texttt{\arrayrulecolor} in order to specify the color of the rules.

With \texttt{nicematrix}, it’s possible to specify the color of the rules even when \texttt{colortbl} is not loaded. For sake of compatibility, the command is also named \texttt{\arrayrulecolor}. The environments of \texttt{nicematrix} also provide a key \texttt{rules/color} to fix the color of the rules in the current environment.

\begin{NiceTabular}{|ccc|}
| \hline
| rose & tulipe & lys \\
| arum & iris & violette \\
| muguet & dahlia & souci \\
| \hline
\end{NiceTabular}

\begin{NiceTabular}{|ccc|}
\hline
rose & tulipe & lys \\
arum & iris & violette \\
muguet & dahlia & souci \\
\hline
\end{NiceTabular}

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command \texttt{\OnlyMainNiceMatrix} described on page 32.

5.3 The keys hlines and vlines

The key \texttt{hlines} draws all the horizontal rules and the key \texttt{vlines} draws all the vertical rules excepted in the blocks (and the virtual blocks determined by dotted lines). In fact, in the environments with delimiters (as \texttt{\pNiceMatrix} or \texttt{\bNiceArray}) the exteriors rules are not drawn (as expected).

\begin{pNiceMatrix}
\begin{NiceTabular}{cccc}
\hline
rose & tulipe & marguerite & dahlia \\
violette & \Block{2-2}{\LARGE\color{blue} fleurs} & & souci \\
pervenche & & & lys \\
arum & iris & jacinthe & muguet
\end{NiceTabular}
\end{pNiceMatrix}

5.4 The key hvlines

The key \texttt{hvlines} draws all the vertical and horizontal rules excepted in the blocks (and the virtual blocks determined by dotted lines).

\begin{NiceTabular}{cccc}
\hline
rose & tulipe & marguerite & dahlia \\
violette & \Block{2-2}{\LARGE\color{blue} fleurs} & souci \\
pervenche & & & lys \\
arum & iris & jacinthe & muguet
\end{NiceTabular}
5.5 The key \texttt{hvlines-except-corners}

The key \texttt{hvlines-except-corners} draws all the horizontal and vertical rules, excepted in the blocks (and the virtual blocks determined by dotted lines) and excepted in the empty corners.

\begin{NiceTabular}{*{6}{c}}[hvlines-except-corners,cell-space-top-limit=3pt]
  & & & & A \\
  & & A & A & A \\
  & & & A \\
  & \Block{2-2}{B} & & A \\
  & & & A \\
  & A & A & A \\
\end{NiceTabular}

\begin{NiceTabular}{*{6}{c}}%[hvlines-except-corners=NE,cell-space-top-limit=3pt]
  & & & & 1 \\
  & & 1 & 1 & 1 \\
  & & 1 & 2 & 1 \\
  & & 1 & 3 & 3 & 1 \\
  & & 1 & 4 & 6 & 4 & 1 \\
  & & 1 & 5 & 10 & 10 & 5 & 1 \\
\end{NiceTabular}

As we can see, an “empty corner” is composed by the reunion of all the empty rectangles starting from the cell actually in the corner of the array.

It’s possible to give as value to the key \texttt{hvlines-except-corners} a list of the corners to take into consideration. The corners are designed by \texttt{NW}, \texttt{SW}, \texttt{NE} and \texttt{SE} (\textit{north west}, \textit{south west}, \textit{north east} and \textit{south east}).

\begin{NiceTabular}{*{5}{c}}[hvlines]
\diagbox{x}{y} & e & a & b & c \\
 e & e & a & b & c \\
 a & a & e & c & b \\
 b & b & c & e & a \\
 c & c & b & a & e \\
\end{NiceTabular}

\begin{NiceArray}[hvlines]
\diagbox{x}{y} & e & a & b & c \\
 e & e & a & b & c \\
 a & a & e & c & b \\
 b & b & c & e & a \\
 c & c & b & a & e \\
\end{NiceArray}

\texttt{It’s possible to use the command \texttt{diagbox} in a \texttt{Block}.}

\textsuperscript{3}The author of this document considers that type of construction as graphically poor.
5.7 Dotted rules

In the environments of the package nicematrix, it’s possible to use the command \hdottedline (provided by nicematrix) which is a counterpart of the classical commands \hline and \hdashline (the latter is a command of arydshln).

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
\hdottedline
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{pNiceMatrix}

\begin{NiceArray}{cccc}
1 & 2 & 3 & 4 & 5 \\
\hdottedline
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{NiceArray}

In the environments with an explicit preamble (like \{NiceTabular}, \{NiceArray}, etc.), it’s possible to draw a vertical dotted line with the specifier “:”.

\begin{NiceArray}{cccc:}
1 & 2 & 3 & 4 & 5 \\
\hdottedline
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{NiceArray}

It’s possible to change in nicematrix the letter used to specify a vertical dotted line with the option letter-for-dotted-lines available in \NiceMatrixOptions.

Remark: In the package array (on which the package nicematrix relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule\(^4\). In nicematrix, the dotted lines drawn by \hdottedline and “:” do likewise.

6 The color of the rows and columns

6.1 Use of colortbl

We recall that the package colortbl can be loaded directly with \usepackage{colortbl} or by loading xcolor with the key table: \usepackage[ table]{xcolor}.

Since the package nicematrix is based on array, it’s possible to use colortbl with nicematrix. However, there are two drawbacks:

- The package colortbl patches array, leading to some incompatibilities (for example with the command \hdotsfor).
- The package colortbl constructs the array row by row, alternating colored rectangles, rules and contents of the cells. The resulting PDF is difficult to interpret by some PDF viewers and may lead to artefacts on the screen.
  - Some rules seem to disappear. This is because many PDF viewers give priority to graphical element drawn posteriorly (which is in the spirit of the “painting model” of PostScript and PDF). Concerning this problem, MuPDF (which is used, for instance, by SumatraPDF) gives better results than Adobe Reader).
  - A thin white line may appear between two cells of the same color. This phenomenon occurs when each cell is colored with its own instruction \texttt{fill} (the PostScript operator \texttt{fill} noted \texttt{f} in PDF). This is the case with colortbl: each cell is colored on its own, even when \texttt{\columncolor} or \texttt{\rowcolor} is used.
  As for this phenomenon, Adobe Reader gives better results than MuPDF.

The package nicematrix provides tools to avoid those problems.

\(^4\)In fact, this is true only for \hline and “|” but not for \cline: cf p. 6
6.2 The tools of nicematrix in the code-before

The package nicematrix provides some tools (independent of colortbl) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.

The extension nicematrix provides a key code-before for some code that will be executed before the drawing of the tabular. In this code-before, new commands are available: \cellcolor, \rectanglecolor, \rowcolor, \columncolor, \rowcolors and \chessboardcolors.

All these commands accept an optional argument (between square brackets and in first position) which is the color model for the specification of the colors.

- The command \cellcolor takes its name from the command \cellcolor of colortbl.
  This command takes in as mandatory arguments a color and a list of cells, each of which with the format \textit{i-j} where \textit{i} is the number of row and \textit{j} the number of column of the cell.

\begin{NiceTabular}{|c|c|c|}[code-before = \cellcolor{red!15}{3-1,2-2,1-3}]
\hline
a & b & c \ \hline
e & f & g \ \hline
h & i & j \ \hline
end{NiceTabular}

A command \cellcolor generates only one instruction \texttt{fill} (coded \texttt{f}) in the resulting PDF.

- The command \rectanglecolor takes three mandatory arguments. The first is the color. The second is the upper-left cell of the rectangle and the third is the lower-right cell of the rectangle.

\begin{NiceTabular}{|c|c|c|}[code-before = \rectanglecolor{blue!15}{2-2}{3-3}]
\hline
a & b & c \ \hline
e & f & g \ \hline
h & i & j \ \hline
end{NiceTabular}

- The command \rowcolor takes its name from the command \rowcolor of colortbl. Its first mandatory argument is the color and the second is a comma-separated list of rows or interval of rows with the form \textit{a-b} (an interval of the form \textit{a-b} represent all the rows from the row \textit{a} until the end).

$\begin{NiceArray}{lll}[hvlines, code-before = \rowcolor{red!15}{1,3-5,8-}]
\textcolor{red!15}{a_1} & b_1 & c_1 \ \\textcolor{red!15}{a_2} & b_2 & c_2 \ \textcolor{red!15}{a_3} & b_3 & c_3 \ \textcolor{red!15}{a_4} & b_4 & c_4 \ \textcolor{red!15}{a_5} & b_5 & c_5 \ \textcolor{red!15}{a_6} & b_6 & c_6 \ \textcolor{red!15}{a_7} & b_7 & c_7 \ \textcolor{red!15}{a_8} & b_8 & c_8 \ \textcolor{red!15}{a_9} & b_9 & c_9 \ \textcolor{red!15}{a_{10}} & b_{10} & c_{10} \ \end{NiceArray}$

A command \rowcolor generates only one instruction \texttt{fill} (coded \texttt{f}) in the resulting PDF.
• The command \columncolor takes its name from the command \columncolor of colortbl. Its syntax is similar to the syntax of \rowcolor.

• The command \rowcolors (with a s) takes its name from the command \rowcolors of xcolor. The s emphasizes the fact that there is two colors. This command colors alternately the rows of the tabular, beginning with the row whose number is given in first (mandatory) argument. The two other (mandatory) arguments are the colors.

\begin{NiceTabular}{@{}lr@{}}[hlines,code-before = \rowcolors{1}{blue!10}{}]
John & 12 \\ Stephens & 8 \\ Sarah & 18 \\ Ashley & 20 \\ Henry & 14 \\ Madison & 15
\end{NiceTabular}

There is a key respect-blocks for the instruction \rowcolors. With that key, the “rows” alternately colored may extend over several rows if they have to incorporate blocks.

\begin{NiceTabular}{lr}[hvlines,code-before = \rowcolors{1}{blue!10}{}[respect-blocks]]
TheBlock{2-1}{John} & 12 \\ & 13 \\ & 8 \\ TheBlock{3-1}{Sarah} & 18 \\ & 17 \\ & 15 \\ Ashley & 20 \\ Henry & 14 \\ TheBlock{2-1}{Madison} & 15 \\ & 19
\end{NiceTabular}

• The command \chessboardcolors takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

$\begin{pNiceMatrix}[r,margin, code-before=\chessboardcolors{red!15}{blue!15}]
1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1
\end{pNiceMatrix}$

We have used the key r which aligns all the columns rightwards (cf. p. 25).

One should remark that these commands are compatible with the commands of booktabs (\toprule, \midrule, \bottomrule, etc).

\footnote{The command \rowcolors of xcolor is available when xcolor is loaded with the option table.}
\begin{NiceTabular}[c]{lSSSS}
\rowcolor{red!15}{1-2} \rowcolors{3}{blue!15}{}
toprule
\Block{2-1}{Product} & \Block{1-3}{dimensions (cm)} & & & \Block{2-1}{\rotate Price} \\
\midrule
& L & l & h & \\
\midrule
small & 3 & 5.5 & 1 & 30 \\
standard & 5.5 & 8 & 1.5 & 50.5 \\
premium & 8.5 & 10.5 & 2 & 80 \\
extra & 8.5 & 10 & 1.5 & 85.5 \\
special & 12 & 12 & 0.5 & 70 \\
\bottomrule
\end{NiceTabular}

We have used the type of column \texttt{S} of \texttt{siunitx}.

### 6.3 Color tools with the syntax of \texttt{colortbl}

It's possible to access the preceding tools with a syntax close to the syntax of \texttt{colortbl}. For that, one must use the key \texttt{colortbl-like} in the current environment.\footnote{As of now, this key is not available in \texttt{NiceMatrixOptions}.}

There are three commands available (they are inspired by \texttt{colortbl} but are independent of \texttt{colortbl}):

- \texttt{cellcolor} which colorizes a cell;
- \texttt{rowcolor} which must be used in a cell and which colorizes the end of the row;
- \texttt{columncolor} which must be used in the preamble of the environment with the same syntax as the corresponding command of \texttt{colortbl} (however, unlike the command \texttt{columncolor} of \texttt{colortbl}, this command \texttt{columncolor} can appear within another command, itself used in the preamble).

\begin{NiceTabular}[colortbl-like]{>{\Blue}c>{\Blue}cc}
\toprule
\rowcolor{red!15}
Last name & First name & Birth day \\
\midrule
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

Each instruction \texttt{cellcolor}, \texttt{rowcolor} or \texttt{columncolor} will generate an instruction \texttt{fill} (coded \texttt{f}) in the resulting PDF. In cases of juxtaposed colored rectangles, one may have a thin
white color line in some PDF viewers\footnote{For example SumatraPDF, which uses MuPDF of Artifex Software, or PDF.js used by Firefox.} (between the two first columns in the above example). In you want to avoid this problem, you should use the tools in the code-before. That’s what we do with the following code.

\begin{NiceTabular}{ccc} %
[\textcolor{blue!15}{colortbl-like}\{ccc\}]%\[\text{code-before = \textcolor{blue!15}{columncolor\{blue!15\}\{1,2\}}}
\toprule
\rowcolor{red!15}
Last name & First name & Birth day \\
\midrule
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

\begin{NiceTabular}{Wc{2cm}cc}[hvlines]
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo
\end{NiceTabular}

In the environments with an explicit preamble (like \{NiceTabular\}, \{NiceArray\}, etc.), it’s possible to fix the width of a given column with the standard letters \texttt{w} and \texttt{W} of the package \texttt{array}.

\begin{NiceMatrix}[
\begin{NiceTabular}{ccc}
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{NiceTabular} %
\begin{NiceMatrix}
\begin{NiceArray}[hvlines]
\begin{NiceTabular}{Wc{2cm}cc}
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo
\end{NiceTabular}
\end{NiceMatrix}

In the environments of \texttt{nicematrix}, it’s also possible to fix the minimal width of all the columns of an array directly with the key \texttt{columns-width}.

\begin{NiceMatrix}[
\begin{NiceTabular}{ccc}
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{NiceTabular} %
\begin{NiceMatrix}
\begin{NiceArray}[hvlines]
\begin{NiceTabular}{Wc{2cm}cc}
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo
\end{NiceTabular}
\end{NiceMatrix}

Note that the space inserted between two columns (equal to 2 \texttt{\tabcolsep} in \{NiceTabular\} and to 2 \texttt{\arraycolsep} in the other environments) is not suppressed (of course, it’s possible to suppress this space by setting \texttt{\tabcolsep} or \texttt{\arraycolsep} equal to 0 pt before the environment).

It’s possible to give the special value \texttt{auto} to the option \texttt{columns-width}: all the columns of the array will have a width equal to the widest cell of the array.\footnote{The result is achieved with only one compilation (but PGF/Tikz will have written informations in the \texttt{.aux} file and a message requiring a second compilation will appear).}

\begin{NiceMatrix}[
\begin{NiceTabular}{ccc}
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{NiceTabular} %
\begin{NiceMatrix}
\begin{NiceArray}[hvlines]
\begin{NiceTabular}{Wc{2cm}cc}
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo
\end{NiceTabular}
\end{NiceMatrix}

\footnote{For example SumatraPDF, which uses MuPDF of Artifex Software, or PDF.js used by Firefox.}
Without surprise, it’s possible to fix the minimal width of the columns of all the matrices of a current scope with the command \NiceMatrixOptions.

\NiceMatrixOptions{columns-width=10mm}
\begin{pNiceMatrix}
a & b \\ c & d \\
\end{pNiceMatrix}
\begin{pNiceMatrix}
1 & 1245 \\ 345 & 2 \\
\end{pNiceMatrix}

But it’s also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment \NiceMatrixBlock with the option auto-columns-width. The environment \NiceMatrixBlock has no direct link with the command \Block presented previously in this document (cf. p. 4).

\begin{NiceMatrixBlock}[auto-columns-width]
\begin{array}{c}
\begin{bNiceMatrix}
9 & 17 \\ -2 & 5 \\
\end{bNiceMatrix} \\
\begin{bNiceMatrix}
1 & 1245345 \\ 345 & 2 \\
\end{bNiceMatrix}
\end{array}
\end{NiceMatrixBlock}

Several compilations may be necessary to achieve the job.

8 The exterior rows and columns

The options first-row, last-row, first-col and last-col allow the composition of exterior rows and columns in the environments of nicematrix.

A potential “first row” (exterior) has the number 0 (and not 1). Idem for the potential “first column”.

\begin{pNiceMatrix}[first-row,last-row,first-col,last-col]
& C_1 & \Cdots & & C_4 & \L_1 \\
& a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
& a_{21} & a_{22} & a_{23} & a_{24} & \Vdots \\
& a_{31} & a_{32} & a_{33} & a_{34} & \L_4 \\
& a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
& C_1 & \Cdots & & C_4 & \L_4
\end{pNiceMatrix}

The dotted lines have been drawn with the tools presented p. 16.

\footnote{At this time, this is the only usage of the environment \NiceMatrixBlock but it may have other usages in the future.}
We have several remarks to do.

- For the environments with an explicit preamble (i.e. \{NiceArray\} and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type \texttt{r} for the first column and \texttt{l} for the last one.

- One may wonder how \texttt{nicematrix} determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
  - For the environments with explicit preamble, like \{NiceTabular\} and \{pNiceArray\}, the number of columns can obviously be computed from the preamble.
  - When the option \texttt{light-syntax} (cf. p. 27) is used, \texttt{nicematrix} has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option \texttt{light-syntax}). The analysis of this whole body gives the number of rows (but not the number of columns).
  - In the other cases, \texttt{nicematrix} compute the number of rows and columns during the first compilation and write the result in the \texttt{aux} file for the next run.

It’s possible to control the appearance of these rows and columns with options \texttt{code-for-first-row}, \texttt{code-for-last-row}, \texttt{code-for-first-col} and \texttt{code-for-last-col}. These options specify tokens that will be inserted before each cell of the corresponding row or column.

\begin{verbatim}
\NiceMatrixOptions{
code-for-first-row = \color{red},
code-for-first-col = \color{blue},
code-for-last-row = \color{green},
code-for-last-col = \color{magenta}}
\begin{pNiceArray}{cc|cc}
[first-row,last-row=5,first-col,last-col,nullify-dots]
  & C_1 & \Cdots & & C_4 & \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
\Vdots & a_{21} & a_{22} & a_{23} & a_{24} & \Vdots \\
\hline
  & a_{31} & a_{32} & a_{33} & a_{34} & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
  & C_1 & \Cdots & & C_4 &
\end{pNiceArray}
\end{verbatim}

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn’t extend in the exterior rows and columns.

  However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command \texttt{\OnlyMainNiceMatrix} described on page 32.

- A specification of color present in \texttt{code-for-first-row} also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to \texttt{xdots/color}). Idem for the other exterior rows and columns.
Logically, the potential option *columns-width* (described p. 13) doesn’t apply to the “first column” and “last column”.

For technical reasons, it’s not possible to use the option of the command `\` after the “first row” or before the “last row” (the placement of the delimiters would be wrong).

## 9 The continuous dotted lines

Inside the environments of the package *nicematrix*, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots`, and `\iddots`.\(^{10}\)

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells\(^{11}\) on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it’s a horizontal line; for `\Vdots`, it’s a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It’s possible to change the color of these lines with the option `color`.\(^{12}\)

```latex
\begin{bNiceMatrix}
a_1 & \Cdots & & & a_1 \\
\Vdots & a_2 & \Cdots & & a_2 \\
& \Vdots & \Ddots[\text{color=red}] \\
\end{bNiceMatrix}
```

In order to represent the null matrix, one can use the following code:

```latex
\begin{bNiceMatrix}
0 & \Cdots & 0 \\
\Vdots & & \Vdots \\
0 & \Cdots & 0 
\end{bNiceMatrix}
```

However, one may want a larger matrix. Usually, in such a case, the users of LaTeX add a new row and a new column. It’s possible to use the same method with *nicematrix*:

```latex
\begin{bNiceMatrix}
0 & \Cdots & \Cdots & 0 \\
\Vdots & & & \Vdots \\
\Vdots & & & \Vdots \\
0 & \Cdots & \Cdots & 0 
\end{bNiceMatrix}
```

In the first column of this example, there are two instructions `\Vdots` but, of course, only one dotted line is drawn.

In fact, in this example, it would be possible to draw the same matrix more easily with the following code:

```latex
\begin{bNiceMatrix}
0 & \Cdots & & 0 \\
\Vdots & & & \Vdots \\
& \Vdots & & \Vdots \\
0 & \Cdots & \Cdots & 0 
\end{bNiceMatrix}
```

\(^{10}\)The command `\Iddots`, defined in *nicematrix*, is a variant of `\ddots` with dots going forward. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of *unicode-math*.

\(^{11}\)The precise definition of a “non-empty cell” is given below (cf. p. 33).

\(^{12}\)It’s also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 19.
There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\hspace*` for the vertical dimension and a command `\hspace` in a cell for the horizontal dimension. However, a command `\hspace*` might interfere with the construction of the dotted lines. That’s why the package `nicematrix` provides a command `\Hspace` which is a variant of `\hspace` transparent for the dotted lines of `nicematrix`.

\begin{bNiceMatrix}
0 & \Cdots & \Hspace*{1cm} & 0 \\
Vdots & & & \Vdots \\
0 & \Cdots & & 0
\end{bNiceMatrix}

9.1 The option nullify-dots

Consider the following matrix composed classically with the environment `{pmatrix}` of `amsmath`.

\[A = \begin{pmatrix}
h & i & j & k & l & m \\
x & & & & & x
\end{pmatrix}\]

If we add `\ldots` instructions in the second row, the geometry of the matrix is modified.

\[B = \begin{pmatrix}
h & i & j & k & l & m \\
x & \ldots & \ldots & \ldots & \ldots & x
\end{pmatrix}\]

By default, with `nicematrix`, if we replace `{pmatrix}` by `{pNiceMatrix}` and `\ldots` by `\Ldots`, the geometry of the matrix is not changed.

\[C = \begin{pNiceMatrix}
\nullify-dots \\
h & i & j & k & l & m \\
x & \Ldots & \Ldots & \Ldots & \Ldots & x
\end{pNiceMatrix}\]

However, one may prefer the geometry of the first matrix `A` and would like to have such a geometry with a dotted line in the second row. It’s possible by using the option `nullify-dots` (and only one instruction `\Ldots` is necessary).

\[D = \begin{pNiceMatrix}[nullify-dots]
\nullify-dots \\
h & i & j & k & l & m \\
x & \Ldots & \Ldots & \Ldots & \Ldots & x
\end{pNiceMatrix}\]

The option `nullify-dots` smashes the instructions `\Ldots` (and the variants) horizontally but also vertically.

9.2 The commands `\hdotsfor` and `\vdotsfor`

Some people commonly use the command `\hdotsfor` of `amsmath` in order to draw horizontal dotted lines in a matrix. In the environments of `nicematrix`, one should use instead `\hdotsfor` in order to draw dotted lines similar to the other dotted lines drawn by the package `nicematrix`. As with the other commands of `nicematrix` (like `\Cdots`, `\Ldots`, `\Vdots`, etc.), the dotted line drawn with `\hdotsfor` extends until the contents of the cells on both sides.

\[\begin{bNiceMatrix}
0 & \hdotsfor{1cm} & 0 \\
\end{bNiceMatrix}\]

\[\begin{bNiceMatrix}
0 & \hdotsfor{1cm} & 0
\end{bNiceMatrix}\]

In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it’s possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`; see p. 13.
However, if these cells are empty, the dotted line extends only in the cells specified by the argument of \Hdotsfor (by design).

\[\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
1 & \Hdotsfor{3} & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5
\end{pNiceMatrix}\]

Remark: Unlike the command \hdotsfor of amsmath, the command \Hdotsfor may be used when the package colortbl is loaded (but you might have problem if you use \rowcolor on the same row as \Hdotsfor).

The package nicematrix also provides a command \Vdotsfor similar to \Hdotsfor but for the vertical dotted lines. The following example uses both \Hdotsfor and \Vdotsfor:

\begin{bNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n] \\
& \Hdotsfor{1} & \Vdotsfor{1} \\
C[a_n,a_1] & \Cdots & C[a_n,a_n]
\end{bNiceMatrix}

9.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of nicematrix without having to modify the code of each matrix. It’s possible with the keys: \texttt{renew-dots} and \texttt{renew-matrix}.\footnote{The options \texttt{renew-dots}, \texttt{renew-matrix} can be fixed with the command \texttt{\NiceMatrixOptions} like the other options. However, they can also be fixed as options of the command \texttt{\usepackage}. There is also a key \texttt{transparent} which is an alias for the conjunction of \texttt{renew-dots} and \texttt{renew-matrix} but it must be considered as obsolete.}
• The option **renew-dots**

With this option, the commands \ldots, \cdots, \vdots, \ddots, \iddots and \hdotsfor are redefined within the environments provided by `nicematrix` and behave like \Ldots, \Cdots, \Vdots, \Ddots, \Iddots and \Hdotsfor; the command \ldots ("automatic dots" of `amsmath`) is also redefined to behave like \Ldots.

• The option **renew-matrix**

With this option, the environment `{matrix}` is redefined and behave like `{NiceMatrix}`, and so on for the five variants.

Therefore, with the keys **renew-dots** and **renew-matrix**, a classical code gives directly the output of `nicematrix`.

```latex
\NiceMatrixOptions{renew-dots, renew-matrix}
\begin{pmatrix}
1 & \cdots & \cdots & 1 \\
0 & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & 1
\end{pmatrix}
```

### 9.4 The labels of the dotted lines

The commands \ldots, \cdots, \Vdots, \Ddots, \Iddots and \Hdotsfor (and the command \line in the code-after which is described p. 21) accept two optional arguments specified by the tokens _ and ^ for labels positioned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

```latex
$\begin{bNiceMatrix}
1 & \hspace*{1cm} & 0 \\
& \text{\textbackslash ddots}^{n \text{ \textbackslash text \ times}} & \\
0 & & 1
\end{bNiceMatrix}$
```

### 9.5 Customization of the dotted lines

The dotted lines drawn by \ldots, \cdots, \Vdots, \Ddots, \Iddots and \Hdotsfor (and by the command \line in the code-after which is described p. 21) may be customized by three options (specified between square brackets after the command):

- **color**;
- **shorten**;
- **line-style**.

These options may also be fixed with `\NiceMatrixOptions` or at the level of a given environment but, in those cases, they must be prefixed by `xdots`, and, thus have for names:

- `xdots/color`;
- `xdots/shorten`;
- `xdots/line-style`.

For the clarity of the explanations, we will use those names.

**The option xdots/color**

The option `xdots/color` fixes the color or the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 14.
The option `xdots/shorten`  
The option `xdots/shorten` fixes the margin of both extremities of the line. The name is derived from the options “shorten >” and “shorten <” of Tikz but one should notice that nicematrix only provides `xdots/shorten`. The initial value of this parameter is 0.3 em (it is recommended to use a unit of length dependent of the current font).

The option `xdots/line-style`  
It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).\footnote{The first reason of this behaviour is that the pdf format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the pdf readers. It’s easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded dots needs a specification of each dot in the pdf file.}

\begin{verbatim}
\tikz \draw [dotted] (0,0) -- (5,0) ;
\end{verbatim}

In order to provide lines with rounded dots in the style of those provided by `\ldots` (at least with the Computer Modern fonts), the package nicematrix embeds its own system to draw a dotted line (and this system uses PGF and not Tikz). This style is called standard and that’s the initial value of the parameter `xdots/line-style`. However (when Tikz is loaded) it’s possible to use for `xdots/line-style` any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tizk pathes (with the exception of “color”, “shorten >” and “shorten <”).

Here is for example a tridiagonal matrix with the style `loosely dotted`:

\begin{verbatim}
$\begin{pNiceMatrix}[nullify-dots,xdots/line-style=loosely dotted]
a & b & 0 & \Cdots & 0 \\
\Vdots & & & & & b \\
0 & \Cdots & & 0 & b & a
\end{pNiceMatrix}$
\end{verbatim}

```
\begin{pNiceMatrix}[nullify-dots,xdots/line-style=loosely dotted]
a & b & 0 & \Cdots & 0 \\
\Vdots & & & & & b \\
0 & \Cdots & & 0 & b & a
\end{pNiceMatrix}$
```

9.6 The dotted lines and the rules  
The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier | in the preamble and by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-corners` are not drawn within the blocks).

\begin{verbatim}
$\begin{bNiceMatrix}[margin,hvlines]
\Block{3-3}<\LARGE>{A} & & & 0 \\
& \hspace*{1cm} & & \Vdots \\
& & & 0 \\
0 & \Cdots& 0 & 0 \\
\end{bNiceMatrix}$
```

```
\begin{bNiceMatrix}[margin,hvlines]
\Block{3-3}<\LARGE>{A} & & & 0 \\
& \hspace*{1cm} & & \Vdots \\
& & & 0 \\
0 & \Cdots& 0 & 0 \\
\end{bNiceMatrix}$
```

10 The code-after  
The option `code-after` may be used to give some code that will be executed after the construction of the matrix.\footnote{There is also a key `code-before` described p. 10.}
A special command, called \texttt{\textbackslash line}, is available to draw directly dotted lines between nodes. It takes two arguments for the two cells to rely, both of the form $i-j$ where $i$ is the number of row and $j$ is the number of column. It may be used, for example, to draw a dotted line between two adjacent cells.

\begin{pNiceMatrix}[code-after=\line{2-2}{3-3}]
I & 0 & \Cdots &0 \\
0 & I & \Ddots &
\Vdots
&
0 &\Cdots & 0 &I \\
\end{pNiceMatrix}

For the legibility of the code, an alternative syntax is provided: it’s possible to give the instructions of the \texttt{\textbackslash code-after} at the end of the environment, after the keyword \texttt{\CodeAfter} (for an example, cf. p. 38). \textbf{New 5.5} Before the version 5.5, it was necessary, in some circumstances, to put the keyword \texttt{\omit} before \texttt{\CodeAfter}. Since version 5.5, one must never put \texttt{\omit}.

11 The notes in the tabulars

11.1 The footnotes

The package \texttt{nicematrix} allows, by using \texttt{footnote} or \texttt{footnotehyper}, the extraction of the notes inserted by \texttt{\footnote} in the environments of \texttt{nicematrix} and their composition in the footpage with the other notes of the document.

If \texttt{nicematrix} is loaded with the option \texttt{footnote} (with \texttt{\usepackage[footnote]{nicematrix}} or with \texttt{\PassOptionsToPackage}), the package \texttt{footnote} is loaded (if it is not yet loaded) and it is used to extract the footnotes.

If \texttt{nicematrix} is loaded with the option \texttt{footnotehyper}, the package \texttt{footnotehyper} is loaded (if it is not yet loaded) ant it is used to extract footnotes.

Caution: The packages \texttt{footnote} and \texttt{footnotehyper} are incompatible. The package \texttt{footnotehyper} is the successor of the package \texttt{footnote} and should be used preferently. The package \texttt{footnote} has some drawbacks, in particular: it must be loaded after the package \texttt{xcolor} and it is not perfectly compatible with \texttt{hyperref}.

11.2 The notes of tabular

The package \texttt{nicematrix} also provides a command \texttt{\tabularnote} which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say \texttt{\NiceTabular}, \texttt{\NiceArray} and \texttt{\NiceMatrix}.

In fact, this command is available only if the extension \texttt{enumitem} has been loaded (before or after \texttt{nicematrix}). Indeed, the notes are composed at the end of the array with a type of list provided by the package \texttt{enumitem}.

\begin{NiceTabular}{@{}llr@{}}
\toprule
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.} & Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.} & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
<table>
<thead>
<tr>
<th>Last name</th>
<th>First name</th>
<th>Birth day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achard</td>
<td>Jacques</td>
<td>June 5, 2005</td>
</tr>
<tr>
<td>Lefebvre</td>
<td>Mathilde</td>
<td>January 23, 1975</td>
</tr>
<tr>
<td>Vanessa</td>
<td>Stephany</td>
<td>October 30, 1994</td>
</tr>
<tr>
<td>Dupont</td>
<td>Chantal</td>
<td>January 15, 1998</td>
</tr>
</tbody>
</table>

\footnote{\textit{Achard} is an old family of the Poitou.}

\footnote{The name Lefebvre is an alteration of the name Lefebure.}
Table 1: Use of `\tabularnote`

<table>
<thead>
<tr>
<th>Last name</th>
<th>First name</th>
<th>Length of life</th>
</tr>
</thead>
<tbody>
<tr>
<td>Churchill</td>
<td>Wiston</td>
<td>91</td>
</tr>
<tr>
<td>Nightingale</td>
<td>Florence</td>
<td>90</td>
</tr>
<tr>
<td>Schoelcher</td>
<td>Victor</td>
<td>89&lt;sup&gt;d&lt;/sup&gt;</td>
</tr>
<tr>
<td>Touchet</td>
<td>Marie</td>
<td>89</td>
</tr>
<tr>
<td>Wallis</td>
<td>John</td>
<td>87</td>
</tr>
</tbody>
</table>

Some text before the notes.

<sup>a</sup> It's possible to put a note in the caption.

<sup>b</sup> Considered as the first nurse of history.

<sup>c</sup> Nicknamed “the Lady with the Lamp”.

<sup>d</sup> The label of the note is overlapping.

11.3 Customisation of the tabular notes

The tabular notes can be customized with a set of keys available in `\NiceMatrixOptions`. The name of these keys is prefixed by `notes`.

- `notes/para`
- `notes/bottomrule`
- `notes/style`
- `notes/label-in-tabular`
- `notes/label-in-list`
- `notes/enumitem-keys`
- `notes/enumitem-keys-para`
- `notes/code-before`

For sake of commodity, it is also possible to set these keys in `\NiceMatrixOptions` via a key `notes` which takes in as value a list of pairs `key=value` where the name of the keys need no longer be prefixed by `notes`:

```
\NiceMatrixOptions
  {
    notes =
    {
      bottomrule,  
      style = ... ,
      label-in-tabular = ... ,
      enumitem-keys =
      {
        labelsep = ... ,
        align = ... ,
        ...
      }
    }
  }
```

We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.
  Initial value: `false`
  That key is also available within a given environment.
• The key `notes/bottomrule` adds a `\bottomrule` of `booktabs` after the notes. Of course, that rule is drawn only if there is really notes in the tabular. The package `booktabs` must have been loaded (before or after the package `nicematrix`). If it is not, an error is raised.

Initial value: `false`

That key is also available within a given environment.

• The key `notes/style` is a command whose argument is specified by `#1` and which gives the style of numerotation of the notes. That style will be used by `\ref` when referencing a tabular note marked with a command `\label`. The labels formatted by that style are used, separated by commas, when the user puts several consecutive commands `\tabularnote`. The marker `#1` is meant to be the name of a LaTeX counter.

Initial value: `\textit{\alph{#1}}`

Another possible value should be a mere `\arabic{#1}`

• The key `notes/label-in-tabular` is a command whose argument is specified by `#1` which is used when formatting the label of a note in the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.

Initial value: `\textsuperscript{#1}`

In French, it’s a tradition of putting a small space before the label of note. That tuning could be acheived by the following code:

```latex
\NiceMatrixOptions{notes/label-in-tabular = \,\textsuperscript{#1}}
```

• The key `notes/label-in-list` is a command whose argument is specified by `#1` which is used when formatting the label in the list of notes at the end of the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.

Initial value: `\textsuperscript{#1}`

In French, the labels of notes are not composed in upper position when composing the notes. Such behaviour could be acheived by:

```latex
\NiceMatrixOptions{notes/label-in-list = #1.\nobreak\hspace{0.25em}}
```

The command `\nobreak` is for the event that the option `para` is used.

• The notes are composed at the end of the tabular by using internally a style of list of `enumitem`.

The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that type of list.

Initial value: `noitemsep , leftmargin = * , align = left , labelsep = 0pt`

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 23).

• The key `notes/enumitem-keys-para` is similar to the previous one but corresponds to the type of list used when the option `para` is in force. Of course, when the option `para` is used, a list of type `inline` (as called by `enumitem`) is used and the pairs `key=value` should correspond to such a list of type `inline`.

Initial value: `afterlabel = \nobreak, itemjoin = \quad`

• The key `notes/code-before` est une token list inserted by `nicematrix` just before the composition of the notes at the end of the tabular.

Initial value: `empty`

For example, if one wishes to compose all the notes in gray and `\footnotesize`, he should use that key:

```latex
\NiceMatrixOptions{notes/code-before = \footnotesize \color{gray}}
```
It’s also possible to add `\raggedright` or `\RaggedRight` in that key (`\RaggedRight` is a command of `ragged2e`).

For an example of customization of the tabular notes, see p. 34.

### 11.4 Use of `{NiceTabular}` with `threeparttable`

If you wish to use the environment `{NiceTabular}` or `{NiceTabular*}` in an environment `{threeparttable}` of the eponymous package, you have to patch the environment `{threeparttable}` with the following code (with a version of LaTeX at least 2020/10/01).

\begin{verbatim}
\makeatletter
\AddToHook{env/threeparttable/begin}{\TPT@hookin{NiceTabular}\TPT@hookin{NiceTabular*}}
\makeatother
\end{verbatim}

### 12 Other features

#### 12.1 Use of the column type S of `siunitx`

If the package `siunitx` is loaded (before or after `nicematrix`), it’s possible to use the S column type of `siunitx` in the environments of `nicematrix`. The implementation doesn’t use explicitly any private macro of `siunitx`.

\begin{verbatim}
\begin{pNiceArray}{ScWc{1cm}c}[nullify-dots,first-row]
\{C_1} & \Cdots & & C_n \\
2.3 & 0 & \Cdots & 0 \\
12.4 & \Vdots & & \Vdots \\
1.45 \\
7.2 & 0 & \Cdots & 0
\end{pNiceArray}
\end{verbatim}

On the other hand, the d columns of the package `dcolumn` are not supported by `nicematrix`.

#### 12.2 Alignment option in `{NiceMatrix}`

The environments without preamble ({`NiceMatrix`}, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.) provide two options `l` and `r` which generate all the columns aligned leftwards (or rightwards).

\begin{verbatim}
\begin{bNiceMatrix}[r]
\cos x & - \sin x \\
\sin x & \cos x
\end{bNiceMatrix}
\end{verbatim}

#### 12.3 The command `\rotate`

The package `nicematrix` provides a command `\rotate`. When used in the beginning of a cell, this command composes the contents of the cell after a rotation of 90° in the direct sense.

In the following command, we use that command in the `code-for-first-row`.
\NiceMatrixOptions%
\code{code-for-first-row = \scriptstyle \rotate \text{image of }},
code{code-for-last-col = \scriptstyle }
\begin{pNiceMatrix}[first-row,last-col=4]
\begin{array}{cccc}
e_1 & e_2 & e_3 & e_1 \\
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\end{array}
\end{pNiceMatrix}
\[ A = \begin{pmatrix}
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\end{pmatrix} \]

If the command \rotate is used in the “last row” (exterior to the matrix), the corresponding elements are aligned upwards as shown below.

\begin{pNiceMatrix}[last-row=4,last-col=4]
\begin{array}{cccc}
e_1 & e_2 & e_3 & e_1 \\
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\text{image of } e_1 & e_2 & e_3 \\
\end{array}
\end{pNiceMatrix}
\[ A = \begin{pmatrix}
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\end{pmatrix} \]

The option small

With the option small, the environments of the package nicematrix are composed in a way similar to the environment \{smallmatrix\} of the package amsmath (and the environments \{psmallmatrix\}, \{bsmallmatrix\}, etc. of the package mathtools).

\begin{bNiceArray}{cccc|c}
\small,
last-col,
code-for-last-col = \scriptscriptstyle,
columns-width = 3mm
\begin{array}{cccc}
1 & -2 & 3 & 4 & 5 \\
0 & 3 & 2 & 1 & 2 & L_2 \gets 2 L_1 - L_2 \\
0 & 1 & 1 & 2 & 3 & L_3 \gets L_1 + L_3 \\
\end{array}
\end{bNiceArray}
\[ \begin{pmatrix}
1 & -2 & 3 & 4 & 5 \\
0 & 3 & 2 & 1 & 2 \\
0 & 1 & 1 & 2 & 3 \\
\end{pmatrix} \]

One should note that the environment \{NiceMatrix\} with the option small is not composed exactly as the environment \{smallmatrix\}. Indeed, all the environments of nicematrix are constructed upon \{array\} (of the package array) whereas the environment \{smallmatrix\} is constructed directly with an \texttt{halign} of TeX.

In fact, the option small corresponds to the following tuning:
- the cells of the array are composed with \scriptstyle;
- \texttt{arraystretch} is set to 0.47;
- \texttt{arraycolsep} is set to 1.45 pt;
- the characteristics of the dotted lines are also modified.

The counters iRow and jCol

In the cells of the array, it’s possible to use the LaTeX counters \texttt{iRow} and \texttt{jCol} which represent the number of the current row and the number of the current column\textsuperscript{17}. Of course, the user must not

\textsuperscript{17} We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.
change the value of these counters which are used internally by `nicematrix`.
In the `code-before` (cf. p. 10) and in the `code-after` (cf. p. 20), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

\begin{pNiceMatrix}% don't forget the 
  \[first-row, \]
  \[first-col, \]
  code-for-first-row = \texttt{\textcolor{blue}{\textbackslash \textbf{alpha\{jCol\}}}}, \]
  code-for-first-col = \texttt{\textcolor{blue}{\textbackslash \textbf{arabic\{iRow\}}} ] \]
  k & k & k & k \ \%
  k 1 & 2 & 3 & 4 \ \\ %
  k 5 & 6 & 7 & 8 \ \\ %
  k 9 & 10 & 11 & 12 %
\end{pNiceMatrix}$

If LaTeX counters called `iRow` and `jCol` are defined in the document by packages other than `nicematrix` (or by the final user), they are shadowed in the environments of `nicematrix`.

The package `nicematrix` also provides commands in order to compose automatically matrices from a general pattern. These commands are \texttt{\textcolor{blue}{\textbackslash AutoNiceMatrix}}, \texttt{\textcolor{blue}{\textbackslash pAutoNiceMatrix}}, \texttt{\textcolor{blue}{\textbackslash bAutoNiceMatrix}}, \texttt{\textcolor{blue}{\textbackslash vAutoNiceMatrix}}, \texttt{\textcolor{blue}{\textbackslash VAutoNiceMatrix}} and \texttt{\textcolor{blue}{\textbackslash BAutoNiceMatrix}}.

These commands take in two mandatory arguments. The first is the format of the matrix, with the syntax `n-p` where `n` is the number of rows and `p` the number of columns. The second argument is the pattern (it’s a list of tokens which are inserted in each cell of the constructed matrix, excepted in the cells of the potential exterior rows and columns).

\[C = \textcolor{blue}{\textbackslash pAutoNiceMatrix}\{3-3\}\{C_{\text{\textcolor{blue}{\textbf{arabic\{iRow\}}}},\text{\textcolor{blue}{\textbf{arabic\{jCol\}}}}}\}$

$$C = \begin{pmatrix}
C_{1,1} & C_{1,2} & C_{1,3} \\
C_{2,1} & C_{2,2} & C_{2,3} \\
C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix}$$

### 12.6 The option light-syntax

The option \texttt{light-syntax} (inspired by the package `spalign`) allows the user to compose the arrays with a lighter syntax, which gives a better legibility of the TeX source.

When this option is used, one should use the semicolon for the end of a row and spaces or tabulations to separate the columns. However, as usual in the TeX world, the spaces after a control sequence are discarded and the elements between curly braces are considered as a whole.

The following example has been composed with XeLaTeX with `unicode-math`, which allows the use of greek letters directly in the TeX source.

\begin{bNiceMatrix}[light-syntax,first-row,first-col]
\{ a \}
\{ a 2 \cos a \}
\{ b \cos a + \cos b \} \%
\{ a \ 2 \cos a \}
\{ a 2 \cos a \}
\{ b \ \cos a + \cos b \} \%
\{ a 2 \cos a \}
\{ a 2 \cos a \}
\{ b \ \cos a + \cos b \} \%
\end{bNiceMatrix}$

It’s possible to change the character used to mark the end of rows with the option `end-of-row`. As said before, the initial value is a semicolon.

When the option \texttt{light-syntax} is used, it is not possible to put verbatim material (for example with the command \texttt{\textbackslash verb}) in the cells of the array.\footnote{The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn’t behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.}
12.7 The environment \{NiceArrayWithDelims\}

In fact, the environment \{pNiceArray\} and its variants are based upon a more general environment, called \{NiceArrayWithDelims\}. The first two mandatory arguments of this environment are the left and right delimiters used in the construction of the matrix. It’s possible to use \{NiceArrayWithDelims\} if we want to use atypical or asymmetrical delimiters.

\[
\begin{NiceArrayWithDelims}
\{\downarrow}{\uparrow}{ccc}[margin]
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{NiceArrayWithDelims}
\]

13 Use of Tikz with nicematrix

13.1 The nodes corresponding to the contents of the cells

The package nicematrix creates a PGF/Tikz node for each (non-empty) cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix (inter alia).

The nodes of a document must have distinct names. That’s why the names of the nodes created by nicematrix contains the number of the current environment. Indeed, the environments of nicematrix are numbered by an internal global counter.

In the environment with the number \( n \), the node of the row \( i \) and column \( j \) has for name \( \text{nm}-n-i-j \).

The command \NiceMatrixLastEnv provides the number of the last environment of nicematrix (for \LaTeX, it’s a “fully expandable” command and not a counter).

However, it’s advisable to use instead the key name. This key gives a name to the current environment. When the environment has a name, the nodes are accessible with the name “name-\( i-j \)” where name is the name given to the array and \( i \) and \( j \) the numbers of row and column. It’s possible to use these nodes with PGF but the final user will probably prefer to use Tikz (which is a convenient layer upon PGF). However, one should remind that nicematrix doesn’t load Tikz by default.

\[
\begin{pNiceMatrix}[name=mymatrix]
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\CodeAfter
\begin{tikzpicture}
  \draw (mymatrix-2-2) circle (2mm) ;
\end{tikzpicture}
\end{pNiceMatrix}
\]

Don’t forget the options remember picture and overlay.

In the code-after, and if Tikz is loaded, the things are easier. One may design the nodes with the form \( i-j \): there is no need to indicate the environment which is of course the current environment.

\[
\begin{pNiceMatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\CodeAfter
\begin{tikzpicture}
  \draw (2-2) circle (2mm) ;
\end{tikzpicture}
\end{pNiceMatrix}
\]

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used: cf. p. 38).

\[
\begin{pmatrix}
a & a+b & a+b+c \\
(a) & a & a+b \\
(a) & a & a
\end{pmatrix}
\]
13.2 The “medium nodes” and the “large nodes”

In fact, the package \texttt{nicematrix} can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option \texttt{create-medium-nodes} and the second ones with the option \texttt{create-large-nodes}.\footnote{There is also an option \texttt{create-extra-nodes} which is an alias for the conjunction of \texttt{create-medium-nodes} and \texttt{create-large-nodes}.}

These nodes are not used by \texttt{nicematrix} by default, and that’s why they are not created by default.

The names of the “medium nodes” are constructed by adding the suffix “-medium” to the names of the “normal nodes”. In the following example, we have underlined the “medium nodes”. We consider that this example is self-explanatory.

\[
\begin{pmatrix}
a & a+b & a+b+c \\
\hline
a & a & a+b \\
\hline
a & a & a \\
\end{pmatrix}
\]

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.\footnote{There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 14).}

\[
\begin{pmatrix}
a & a+b & a+b+c \\
\hline
a & a & a+b \\
\hline
a & a & a \\
\end{pmatrix}
\]

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options \texttt{left-margin} and \texttt{right-margin} to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options \texttt{left-margin} and \texttt{right-margin}.\footnote{The options \texttt{left-margin} and \texttt{right-margin} take dimensions as values but, if no value is given, the default value is used, which is \texttt{arraycolsep} (by default: 5 pt). There is also an option \texttt{margin} to fix both \texttt{left-margin} and \texttt{right-margin} to the same value.}

\[
\begin{pmatrix}
a & a+b & a+b+c \\
\hline
a & a & a+b \\
\hline
a & a & a \\
\end{pmatrix}
\]

It’s also possible to add more space on both side of the array with the options \texttt{extra-left-margin} and \texttt{extra-right-margin}. These margins are not incorporated in the “large nodes”. It’s possible to fix both values with the option \texttt{extra-margin} and, in the following example, we use \texttt{extra-margin} with the value 3 pt.

\[
\begin{pmatrix}
a & a+b & a+b+c \\
\hline
a & a & a+b \\
\hline
a & a & a \\
\end{pmatrix}
\]

\textbf{Be careful:} These nodes are reconstructed from the contents of the contents cells of the array. Usually, they do not correspond to the cells delimited by the rules (if we consider that these rules are drawn).

Here is an array composed with the following code:

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]

\[
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
\hline
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnon \\
\end{NiceTabular}
\]
Here, we have colored all the cells of the array with \texttt{\chessboardcolors}.

Here are the “large nodes” of this array (without use of \texttt{margin} nor \texttt{extra-margin}).

13.3 The “row-nodes” and the “col-nodes”

The package \texttt{nicematrix} creates a PGF/Tikz node indicating the potential position of each horizontal rule (with the names \texttt{row-i}) and each vertical rule (with the names \texttt{col-j}), as described in the following figure. These nodes are available in the \texttt{code-before} and the \texttt{code-after}.

If we use Tikz (we remind that \texttt{nicematrix} does not load Tikz by default), we can access (in the \texttt{code-before} and the \texttt{code-after}) to the intersection of the horizontal rule \texttt{i} and the vertical rule \texttt{j} with the syntax \texttt{(row-i-|col-j)}.

\[
\begin{NiceMatrix}
\begin{code-before}
\begin{tikz}
\draw [fill = red!15]
(row-7-|col-4) -- (row-8-|col-4) -- (row-8-|col-5) --
(row-9-|col-5) -- (row-9-|col-6) |- cycle ;
\end{tikz}
\end{code-before}
\begin{NiceCode}[command=	exttt{\end{NiceMatrix}}]
\end{NiceCode}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{code-before}
\begin{tikz}
\draw [fill = red!15]
(row-7-|col-4) -- (row-8-|col-4) -- (row-8-|col-5) --
(row-9-|col-5) -- (row-9-|col-6) |- cycle ;
\end{tikz}
\end{code-before}
\begin{NiceCode}[command=	exttt{\end{NiceMatrix}}]
\end{NiceCode}
\end{NiceMatrix}
\] 

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1

30
14 API for the developpers

The package \texttt{nicematrix} provides two variables which are internal but public\textsuperscript{22}:

\begin{itemize}
  \item \texttt{\g_nicematrix_code_before_tl} ;
  \item \texttt{\g_nicematrix_code_after_tl} .
\end{itemize}

These variables contain the code of what we have called the “code-before” and the “code-after”. The developer can use them to add code from a cell of the array (the affection must be global, allowing to exit the cell, which is a TeX group).

One should remark that the use of \texttt{\g_nicematrix_code_before_tl} needs one compilation more (because the instructions are written on the aux file to be used during the next run).

\textit{Example} : We want to write a command \texttt{\hatchcell} to hatch the current cell (with an optional argument between brackets for the color). It’s possible to program such command \texttt{\hatchcell} as follows, explicitely using the public variable \texttt{\g_nicematrix_code_before_tl} (this code requires the Tikz library \texttt{patterns}).

\begin{Verbatim}
\ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatchcell:nnn { \begin { tikzpicture } \fill [ pattern = north\text{-}west\text{-}lines , pattern\text{-}color = #3 ] ( row \#1 \mid col \#2) rectangle ( row \#1 \mid col \#2 ) ; \end { tikzpicture } \}
\NewDocumentCommand \hatchcell { ! O { black } } { \tl_gput_right:Nx \g_nicematrix_code_before_tl \ \{ \__pantigny_hatchcell:nnn \{ \int_use:c { c@iRow } \} \{ \int_use:c { c@jCol } \} \{ #1 \} \} \\ExplSyntaxOff
\end{Verbatim}

Here is an example of use:
\begin{Verbatim}
\begin{NiceTabular}{ccc}[hvlines]
Tokyo & Paris & London \\
Roma & \hatchcell[blue!30]Oslo & Miami \\
Los Angeles & Madrid & Roma
\end{NiceTabular}
\end{Verbatim}

\textsuperscript{22}According to the LaTeX conventions, each variable with name beginning with \texttt{\g_nicematrix} or \texttt{\l_nicematrix} is public and each variable with name beginning with \texttt{\g__nicematrix} or \texttt{\l__nicematrix} is private.
15 Technical remarks

15.1 Definition of new column types

The package \nicematrix provides the command \OnlyMainNiceMatrix which is meant to be used in definitions of new column types. Its argument is evaluated if and only if we are in the main part of the array, that is to say not in an potential exterior row.

For example, one may wish to define a new column type \( ? \) in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job:\(^{23}\):

\[
\newcolumntype{?}{!{\OnlyMainNiceMatrix{\vrule width 1 pt}}}
\]

The heavy vertical rule won’t extend in the exterior rows.\(^{24}\)

\[
\begin{array}{cc?cc}
C_1 & C_2 & C_3 & C_4 \\
\hline
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{array}
\]

This specifier \( ? \) may be used in the standard environments \{tabular\} and \{array\} (of the package array) and, in this case, the command \OnlyMainNiceMatrix is no-op.

15.2 Diagonal lines

By default, all the diagonal lines\(^{25}\) of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions \Ddots in the array can have a marked effect on the final result.

In the following examples, the first \Ddots instruction is written in color:

Example with parallelization (default):

\[
\begin{array}{cc?cc}
C_1 & C_2 & C_3 & C_4 \\
\hline
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{array}
\]

It’s possible to turn off the parallelization with the option parallelize-diags set to false:

The same example without parallelization:

\[
\begin{array}{cc?cc}
C_1 & C_2 & C_3 & C_4 \\
\hline
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{array}
\]

\(^{23}\)The command \vrule is a \TeX (and not \LaTeX) command.
\(^{24}\)Of course, such rule is defined by the classical technics of \nicematrix and, for this reason, won’t cross the double rules of \hline\hline.
\(^{25}\)We speak of the lines created by \Ddots and not the lines created by a command \line in code-after.
It’s possible to specify the instruction \Ddots which will be drawn first (and which will be used to draw the other diagonal dotted lines when the parallelization is in force) with the key draw-first: \Ddots[draw-first].

15.3 The “empty” cells

An instruction like \Ldots, \Cdots, etc. tries to determine the first non-empty cells on both sides. However, an empty cell is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands &). Indeed, a cell which only contains \hspace*{1cm} may be considered as empty.

For nicematrix, the precise rules are as follow.

- An implicit cell is empty. For example, in the following matrix:

\begin{pmatrix}
  a & b \\
  c \\
\end{pmatrix}

the last cell (second row and second column) is empty.

- Each cell whose TeX output has a width equal to zero is empty.

- A cell with a command \Hspace (or \Hspace*) is empty. This command \Hspace is a command defined by the package nicematrix with the same meaning as \hspace except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with nicematrix.

15.4 The option exterior-arraycolsep

The environment \{array\} inserts an horizontal space equal to \arraycolsep before and after each column. In particular, there is a space equal to \arraycolsep before and after the array. This feature of the environment \{array\} was probably not a good idea\(^\text{26}\). The environment \{matrix\} of amsmath and its variants \{pmatrix\}, \{vmatrix\}, etc. of amsmath prefer to delete these spaces with explicit instructions \hskip -\arraycolsep\(^\text{27}\). The package nicematrix does the same in all its environments, \{NiceArray\} included. However, if the user wants the environment \{NiceArray\} behaving by default like the environment \{array\} of array (for example, when adapting an existing document) it’s possible to control this behaviour with the option exterior-arraycolsep, set by the command \NiceMatrixOptions. With this option, exterior spaces of length \arraycolsep will be inserted in the environments \{NiceArray\} (the other environments of nicematrix are not affected).

15.5 Incompatibilities

The package nicematrix is not fully compatible with the package arydshln (because this package redefines many internal of array).

\(^{26}\)In the documentation of \{amsmath\}, we can read: The extra space of \arraycolsep that array adds on each side is a waste so we remove it \{in \{matrix\}\} (perhaps we should instead remove it from array in general, but that’s a harder task).

\(^{27}\)And not by inserting \@{} on both sides of the preamble of the array. As a consequence, the length of the \hline is not modified and may appear too long, in particular when using square brackets.
16 Examples

16.1 Notes in the tabulars

The tools provided by \nicematrix for the composition of the tabular notes have been presented in the section 11 p. 21.

Let’s consider that we wish to number the notes of a tabular with stars.\footnote{Of course, it’s realistic only when there is very few notes in the tabular.}

First, we write a command \texttt{\stars} similar the well-known commands \texttt{\arabic}, \texttt{\alph}, \texttt{\Alph}, etc. which produces a number of stars equal to its argument\footnote{In fact: the value of its argument.}

\begin{verbatim}
\ExplSyntaxOn
\NewDocumentCommand \stars { m }
 { \prg_replicate:nn { \value { #1 } } { \star } }
\ExplSyntaxOff
\end{verbatim}

Of course, we change the style of the labels with the key \texttt{notes/style}. However, it would be interesting to change also some parameters in the type of list used to compose the notes at the end of the tabular. First, we required a composition flush right for the labels with the setting \texttt{align=right}. Moreover, we want the labels to be composed on a width equal to the width of the widest label. The widest label is, of course, the label with the greatest number of stars. We know that number: it is equal to \texttt{\value{tabularnote}} (because \texttt{tabularnote} is the LaTeX counter used by \texttt{\tabularnote} and, therefore, at the end of the tabular, its value is equal to the total number of tabular notes). We use the key \texttt{widest*=} of \texttt{enumitem} in order to require a width equal to that value: \texttt{widest*=\value{tabularnote}}.

\begin{verbatim}
\NiceMatrixOptions
 { notes =
 { style = \stars{#1} ,
    enumitem-keys =
    { widest* = \value{tabularnote} ,
      align = right
    }
  }
}
\end{verbatim}

\begin{NiceTabular}{{}llr{}}[first-row,code-for-first-row = \bfseries]
\toprule
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.} & Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.} & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
<table>
<thead>
<tr>
<th>Last name</th>
<th>First name</th>
<th>Birth day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achard</td>
<td>Jacques</td>
<td>June 5, 2005</td>
</tr>
<tr>
<td>Lefebvre</td>
<td>Mathilde</td>
<td>January 23, 1975</td>
</tr>
<tr>
<td>Vanesse</td>
<td>Stephany</td>
<td>October 30, 1994</td>
</tr>
<tr>
<td>Dupont</td>
<td>Chantal</td>
<td>January 15, 1998</td>
</tr>
</tbody>
</table>

*Achard is an old family of the Poitou.

**The name Lefebvre is an alteration of the name Lefebure.

### 16.2 Dotted lines

A permutation matrix (as an example, we have raised the value of \texttt{xdots/shorten}).

```latex
\begin{pNiceMatrix}[\texttt{xdots/shorten=0.6em}]
0 & 1 & 0 & \Cdots & 0 \\ 
\Vdots & & & \Ddots & \Vdots \\ 
& & & \Ddots & & \\ 
& & & \Ddots & & 0 \\ 
0 & 0 & & & & 1 \\ 
1 & 0 & \Cdots & & & 0 \\
\end{pNiceMatrix}
```

An example with \texttt{\iddots} (we have raised again the value of \texttt{xdots/shorten}).

```latex
\begin{pNiceMatrix}[\texttt{xdots/shorten=0.9em}]
1 & \Cdots & & 1 \\ 
\Vdots & & & 0 \\ 
& & \iddots & \iddots & \iddots & \Vdots \\ 
1 & 0 & \Cdots & & 0 \\
\end{pNiceMatrix}
```

An example with \texttt{\multicolumn}:

```latex
\begin{BNiceMatrix}[\texttt{nullify-dots}]
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\Cdots & & \multicolumn{6}{C}{10 \text{ other rows}} & \Cdots \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{BNiceMatrix}
```

\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
\end{verbatim}
\end{verbatim}
```

\[\begin{array}{ll}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\ldots & \ldots & 10 \text{ other rows} & \ldots & \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}\]
An example with \Hdotsfor:

\begin{pNiceMatrix}[nullify-dots]
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
\Vdots & \Hdotsfor{4} & \Vdots \\
& \Hdotsfor{4} & & \\
& \Hdotsfor{4} & & \\
& \Hdotsfor{4} & & \\
0 & 1 & 1 & 1 & 1 & 0
\end{pNiceMatrix}

\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}

An example for the resultant of two polynomials:

\begin{vNiceArray}{cccc:ccc}[columns-width=6mm]
\[a_0 & & & b_0 & & \]
\[a_1 & \Ddots & & b_1 & \Ddots & \\
\Vdots & \Ddots & & \Vdots & \Ddots & b_0 \]
\[a_p & & a_0 & \& b_1 & & \\
& \Ddots & a_1 & b_q & \& \Vdots \]
\[& \& \Vdots & \& \Ddots & \& \\
& \& a_p & \& b_q \]
\end{vNiceArray}

\begin{eqnarray*}
\begin{array}{cccc}
a_0 & b_0 & \cdots & a_p \\
a_1 & b_1 & \cdots & a_p \\
\vdots & \vdots & \vdots & \vdots \\
a_p & b_q & \vdots & \vdots \\
\end{array}
\end{eqnarray*}

An example for a linear system:

\begin{pNiceArray}[*6c|c][nullify-dots, last-col, code-for-last-col=\scriptstyle]
1 & 1 & 1 & \Cdots & & 1 & 0 & \& \\
0 & 1 & 0 & \Cdots & & 0 & & L_2 \gets L_2-L_1 \\
0 & 0 & 1 & \Cdots & & \Vdots & & L_3 \gets L_3-L_1 \\
& \& \& \& \& \& \& \& \& \\
\Vdots & & \& \& \& \& \& \& \\
0 & \& \& \& \& \& \& \& L_n \gets L_n-L_1
\end{pNiceArray}
16.3 Dotted lines which are no longer dotted

The option `line-style` controls the style of the lines drawn by `\Ldots`, `\Cdots`, etc. Thus, it’s possible with these commands to draw lines which are not longer dotted.

\begin{NiceMatrixOptions}
  {nullify-dots,code-for-first-col = \color{blue},code-for-first-col=\color{blue}}
\end{NiceMatrixOptions}

$\begin{pNiceMatrix}[first-row,first-col]
\Ldots[^{n \text{ columns}}] \\
1 & 1 & 1 & & 1 \\
1 & 1 & 1 & & 1 \\
\Vdots_{n \text{ rows}} & 1 & 1 & & 1 \\
1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$

16.4 Width of the columns

In the following example, we use `{NiceMatrixBlock}` with the option `auto-columns-width` because we want the same automatic width for all the columns of the matrices.

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
  { last-col,code-for-last-col = \color{blue}\scriptstyle,light-syntax}
\setlength{\extrarowheight}{1mm}
$\begin{pNiceArray}{cccc:c}
1 & 1 & 1 & 1 & {} \\
2 & 4 & 8 & 16 & 9 \\
3 & 9 & 27 & 81 & 36 \\
4 & 16 & 64 & 256 & 100
\end{pNiceArray}$

\begin{pNiceArray}{cccc:c}
1 & 1 & 1 & 1 & {} \\
0 & 2 & 6 & 14 & 7 \\
0 & 6 & 24 & 78 & 33 \\
0 & 12 & 60 & 252 & 96
\end{pNiceArray}$

$...$
\end{NiceMatrixBlock}
16.5 How to highlight cells of the matrix

The following examples require Tikz (by default, nicematrix only loads PGF) and the Tikz library fit. The following lines in the preamble of your document do the job:

```
\usepackage{tikz}
\usetikzlibrary{fit}
```

In order to highlight a cell of a matrix, it’s possible to “draw” one of the correspondant nodes (the “normal node”, the “medium node” or the “large node”). In the following example, we use the “large nodes” of the diagonal of the matrix (with the Tikz key “name suffix”, it’s easy to use the “large nodes”).

We redraw the nodes with other nodes by using the Tikz library fit. Since we want to redraw the nodes exactly, we have to set `inner sep = 0 pt` (if we don’t do that, the new nodes will be larger than the nodes created by nicematrix).

```
\begin{pNiceArray}{>{\strut}cccc}
\{create-large-nodes,margin,extra-margin = 2pt\}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44} \\
\CodeAfter
\begin{tikzpicture}\[name suffix = -large, every node/.style = {draw, inner sep = 0 pt}]
\node [fit = (1-1)] {} ;
\node [fit = (2-2)] {} ;
\node [fit = (3-3)] {} ;
\node [fit = (4-4)] {} ;
\end{tikzpicture}\end{pNiceArray}
```

We should remark that the rules we have drawn are drawn after the construction of the array and thus, they don’t spread the cells of the array. We recall that, on the other side, the command `\hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.\footnote{For the command `\cline`, see the remark p. 6.}
It’s possible to color a row with \texttt{\rowcolor} in the \texttt{code-before} (or with \texttt{\rowcolor} of colortbl in the first cell of the row). However, it’s not possible to do a fine tuning. That’s why we describe now method to highlight a row of the matrix. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library \texttt{fit}. This Tikz node is filled after the construction of the matrix. In order to see the text \textit{under} this node, we have to use transparency with the \texttt{blend mode} equal to \texttt{multiply}.

\begin{verbatim}
\tikzset{highlight/.style={rectangle, 
  fill=red!15, 
  blend mode = multiply, 
  rounded corners = 0.5 mm, 
  inner sep=1pt, 
  fit = #1}}

$\begin{bNiceMatrix}[\text{code-after} = \{\tikz \node [highlight = (2-1) (2-3)] {} \};]$
0 & \Cdots & 0 \\
1 & \Cdots & 1 \\
0 & \Cdots & 0
\end{bNiceMatrix}$

We recall that, for a rectangle of merged cells (with the command \texttt{\Block}), a Tikz node is created for the set of merged cells with the name \texttt{i-j-block} where \texttt{i} and \texttt{j} are the number of the row and the number of the column of the upper left cell (where the command \texttt{\Block} has been issued). If the user has required the creation of the medium nodes, a node of this type is also created with a name suffixed by \texttt{-medium}.

\begin{verbatim}
$\begin{pNiceMatrix}[\text{margin,create-medium-nodes}]$
\Block{3-3}<\Large>{A} & & & 0 \\
\hspace*{1cm} & & & \Vdots \\
\hspace*{1cm} & & & 0 \\
0 & \Cdots& 0 & 0
\CodeAfter
\tikz \node [highlight = (1-1-block-medium)] {} ;$
\end{pNiceMatrix}$

Consider now the following matrix which we have named \texttt{example}.

\begin{verbatim}
$\begin{pNiceArray}{ccc}[\text{name=example,\text{last-col,create-medium-nodes}]$
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray}$

If we want to highlight each row of this matrix, we can use the previous technique three times.
We obtain the following matrix.

\[
\begin{pmatrix}
  a & a + b & a + b + c \\
  a & a & a + b \\
  a & a & a \\
\end{pmatrix}
\]

The result may seem disappointing. We can improve it by using the “medium nodes” instead of the “normal nodes”.

We obtain the following matrix.

\[
\begin{pmatrix}
  a & a + b & a + b + c \\
  a & a & a + b \\
  a & a & a \\
\end{pmatrix}
\]

In the following example, we use the “large nodes” to highlight a zone of the matrix.

\[
\begin{pmatrix}
  A_{11} & A_{12} & A_{13} & A_{14} \\
  A_{21} & A_{22} & A_{23} & A_{24} \\
  A_{31} & A_{32} & A_{33} & A_{34} \\
  A_{41} & A_{42} & A_{43} & A_{44} \\
\end{pmatrix}
\]
16.6 Direct use of the Tikz nodes

In the following example, we illustrate the mathematical product of two matrices.

The use of \{NiceMatrixBlock\} with the option \texttt{auto-columns-width} gives the same width for all the columns and, therefore, a perfect alignment of the two superposed matrices.

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions{nullify-dots}

The three matrices will be displayed using an environment \{array\} (an environment \{tabular\} may also be possible).

\$\begin{array}{cc}
\end{array}\$  

The matrix $B$ has a “first row” (for $C_j$) and that’s why we use the key \texttt{first-row}.

\begin{bNiceArray}{c>{\strut}cccc}[name=B,first-row]
& C_j \\
b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
\Vdots & & \Vdots & & \Vdots \\
& & b_{kj} \\
& & \Vdots \\
b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn}  
\end{bNiceArray} \ \ \ 

The matrix $A$ has a “first column” (for $L_i$) and that’s why we use the key \texttt{first-col}.

\begin{bNiceArray}{cc>{\strut}ccc}[name=A,first-col]
& a_{11} & \Cdots & & & a_{1n} \\
& \Vdots & & & & \Vdots \\
L_i & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} \\
& \Vdots & & & & \Vdots \\
& a_{n1} & \Cdots & & & a_{nn}  
\end{bNiceArray}

In the matrix product, the two dotted lines have an open extremity.

\begin{bNiceArray}{cc>{\strut}ccc}
& & & & \\
& & \Vdots \\
\Cdots & & c_{ij} \\
\end{bNiceArray} 

\begin{tikzpicture}[remember picture, overlay]
\node [highlight = (A-3-1) (A-3-5)] {} ;
\node [highlight = (B-1-3) (B-5-3)] {} ;
\draw [color = gray] (A-3-3) to [bend left] (B-3-3) ;
\end{tikzpicture}
17 Implementation

By default, the package nicematrix doesn't patch any existing code.
However, when the option renew-dots is used, the commands \cdots, \ldots, \vdots, \ddots and \iddots are redefined in the environments provided by nicematrix as explained previously.
In the same way, if the option renew-matrix is used, the environment \texttt{matrix} of \texttt{amsmath} is redefined.

On the other hand, the environment \{array\} is never redefined.
Of course, the package nicematrix uses the features of the package array. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent: the package nicematrix relies upon the fact that the package \{array\} uses \texttt{\textbackslash align} to begin the \texttt{\textbackslash halign}.

Declaration of the package and packages loaded

The prefix nicematrix has been registered for this package.
See: http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf
<@@=nicematrix>

First, we load \texttt{pgfcore} and the module \texttt{shapes}. We do so because it’s not possible to use \texttt{\usepgfmodule} in \texttt{\ExplSyntaxOn}.

\begin{verbatim}
\RequirePackage{pgfcore}
\usepgfmodule{shapes}
\end{verbatim}

We give the traditional declaration of a package written with \texttt{exp13}:

\begin{verbatim}
\RequirePackage{13keys2e}
\ProvidesExplPackage
\nicematrix
\{myfiledate}
\{myfileversion\}
\{Enhanced arrays with the help of PGF/TikZ\}
\end{verbatim}

The command for the treatment of the options of \texttt{\usepackage} is at the end of this package for technical reasons.

We load some packages.

\begin{verbatim}
\RequirePackage { array }
\RequirePackage { amsmath }
\RequirePackage { sparse }
\end{verbatim}
Technical definitions

In some constructions, we will have to use a \{pgfpicture\} which must be replaced by a \{tikzpicture\} if Tikz is loaded. However, this switch between \{pgfpicture\} and \{tikzpicture\} can’t be done dynamically with a conditional because, when the Tikz library external is loaded by the user, the pair \begin{tikzpicture}-\end{tikzpicture} (or \begin{pgfpicture}-\end{pgfpicture}) must be statically “visible” (even when externalization is not activated).

That’s why we create \c_@@_pgfortikzpicture_tl and \c_@@_endpgfortikzpicture_tl which will be used to construct in a \AtBeginDocument the correct version of some commands.

We test whether the current class is revtex4-1 or revtex4-2 because these classes redefines \array (of array) in a way incompatible with our programation.

We define a command \iddots similar to \ddots (\ldots) but with dots going forward (\ldots). We use \ProvideDocumentCommand of \xparse, and so, if the command \iddots has already been defined (for example by the package mathdots), we don’t define it again.

43
This definition is a variant of the standard definition of \ddots.

In the aux file, we will have the references of the PGF/Tikz nodes created by nicematrix. However, when booktabs is used, some nodes (more precisely, some row nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine \pgfutil@check@rerun in the aux file.

\AtBeginDocument
{ 
\@ifpackageloaded { booktabs }
{ \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
{ }
}

The new version of \pgfutil@check@rerun will not check the PGF nodes whose names start with nm- (which is the prefix for the nodes creates by nicematrix).

\bool_new:N \c_@@_colortbl_loaded_bool
\AtBeginDocument
{ 
\@ifpackageloaded { colortbl }
{ \bool_set_true:N \c_@@_colortbl_loaded_bool }
}

The command \CT@arc@ is a command of colortbl which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if colortbl is not loaded. Idem for \CT@drsc@.

\CS@arc@ { }
\CS@arc@ \#1 \#2
\CS@drsc@ { }
\CS@drsc@ \#1 \#2

Idem for \CT@drsc@.

We have to redefine \cline for several reasons. The command \@@_cline will be linked to \cline in the beginning of {NiceArrayWithDelims}. The following commands must not be protected.

\cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
\cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
\int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
\int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
\multispan { \int_eval:n { #2 - #1 + 1 } }
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim
\leaders \hrule \@height \arrayrulewidth \hfill

The following \skip_horizontal:N \c_zero_dim is to prevent a potential \unskip to delete the \leaders\skip_horizontal:N \c_zero_dim

Our \everycr has been modified. In particular, the creation of the row node is in the \everycr (maybe we should put it with the incrementation of \c@iRow). Since the following \cr correspond to a “false row”, we have to nullify \everycr.
\cr\noalign { \skip_vertical:N -\arrayrulewidth }
\cr
\noalign { \skip_vertical:N \c_zero_dim

The following version of \cline spreads the array of a quantity equal to \arrayrulewidth as does \hline. It will be loaded excepted if the key standard-cline has been used.
\cs_set:Npn \@@_cline

We have to act in a fully expandable way since there may be \noalign (in the \multispan) to detect. That’s why we use \@@_cline_i:en.
\cs_set:Npn \@@_cline_i:en
\int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
\int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
\multispan { \int_eval:n { #2 - #1 + 1 } }
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim

The command \cline_i:nn has two arguments. The first is the number of the current column (it must be used in that column). The second is a standard argument of \cline of the form i-j.
\cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
\cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
\int_compare:nNnT { #1 } < { #2 }
\int_compare:nNnT { #2 } > { #1 }
\int_compare:nNnT { #3 } > { #2 + 1 }
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim

Now, #1 is the number of the current column and we have to draw a line from the column #2 to the column #3 (both included).
\int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
\int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
\multispan { \int_eval:n { #2 - #1 } } \& }
\multispan { \int_eval:n { #3 - #2 + 1 } }
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim

You look whether there is another \cline to draw (the final user may put several \cline).

\textsuperscript{31}See question 99041 on TeX StackExchange.
The following commands are only for efficiency. They must *not* be protected because it will be used (for instance) in names of PGF nodes.

\begin{verbatim}
\cs_new:Npn \@@_succ:n #1 { \the \numexpr #1 + 1 \relax }
\cs_new:Npn \@@_pred:n #1 { \the \numexpr #1 - 1 \relax }
\end{verbatim}

The following command is a small shortcut.

\begin{verbatim}
\cs_new:Npn \@@_math_toggle_token: { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }
\cs_new_protected:Npn \@@_set_CT@arc@: { \peek_meaning:NTF \[ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
\cs_new_protected:Npn \@@_set_CT@arc@_i: \[ #1 \] \q_stop { \cs_set:Npn \CT@arc@ { \color \[ #1 \] { #2 } } }
\cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop { \cs_set:Npn \CT@arc@ { \color { #1 } } }
\end{verbatim}

The column S of siunitx

We want to know whether the package siunitx is loaded and, if it is loaded, we redefine the S columns of siunitx.

\begin{verbatim}
\bool_new:N \c_@@_siunitx_loaded_bool
\AtBeginDocument{
\bool_ifpackageloaded { siunitx }{
\bool_set_true:N \c_@@_siunitx_loaded_bool
}{
}
\end{verbatim}

The command \NC@rewrite@S is a LaTeX command created by siunitx in connection with the S column. In the code of siunitx, this command is defined by:
\begin{verbatim}
\renewcommand*{\NC@rewrite@S}[1][ ]{
\@temptokena \exp_after:wN
\tex_the:D \@temptokena
> { \_\_siunitx_table_collect_begin: S {#1} }
< { \_\_siunitx_table_print: }
\NC@find
}
\end{verbatim}

We want to patch this command (in the environments of nicematrix) in order to have:
\begin{verbatim}
\renewcommand*{\NC@rewrite@S}[1][ ]{
\@temptokena \exp_after:wN
\tex_the:D \@temptokena
> { \@@Cell: \_\_siunitx_table_collect_begin: S {#1} }
\@@true_c:
< { \_\_siunitx_table_print: \@@end_Cell: }
\NC@find
}
\end{verbatim}
However, we don’t want to use explicitly any private command of siunitx. That’s why we will extract the name of the two \_\_\_siunitx\_\_\_ commands by their position in the code of \NC@rewrite\@S. Since the command \NC@rewrite\@S appends some tokens to the toks list \@temptokena, we use the \LaTeX\ command \NC@rewrite\@S in a group (\group@begin:–\group@end:) and we extract the two command names which are in the toks \@temptokena. However, this extraction can be done only when siunitx is loaded (and it may be loaded after nicematrix) and, in fact, after the beginning of the document — because some instructions of siunitx are executed in a \AtBeginDocument. That’s why this extraction will be done only at the first use of an environment of nicematrix with the command \@@_adapt\_\_S\_column:.

\begin{verbatim}
164 \cs_set_protected:Npn \@@_adapt\_\_S\_column: {
165 \bool_if:NT \c_@@_siunitx\_loaded_bool {
166 \group@begin:
167 \@temptokena = { }
168 \prg_do_nothing:
169 \NC@rewrite\@S { }
170 \cs_set_eq:NN \NC@find \prg_do_nothing:
171 \NC@rewrite\@S { }
172 Conversion of the toks \@temptokena in a token list of expl3 (the toks are not supported by expl3 but we can, nevertheless, use the option V for \tl_gset:NV).
173 \tl_gset:NV \g_tmpa_tl \@temptokena
174 \group@end:
175 \tl_new:N \c_@@_table\_collect\_begin_tl
176 \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
177 \tl_gset:Nx \c_@@_table\_collect\_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
178 \tl_new:N \c_@@_table\_print_tl
179 \tl_gset:Nx \c_@@_table\_print_tl { \tl_item:Nn \g_tmpa_tl \{-1\} }
180 The token lists \c_@@_table\_collect\_begin_tl and \c_@@_table\_print_tl contain now the two commands of siunitx.
181 If the adaptation has been done, the command \@@_adapt\_\_S\_column: becomes no-op (globally).
182 \}
183 \}
184 \}
185 \}
186 \}
187 \}
188 \}
189 \}
190 \}
191 \}
192 \}
193 \}
194 \}
195 \}
196 \}
197 \}
198 \}
199 \}
200 \}
201 \}
202 \}
\end{verbatim}

The command \@@_renew\_\NC@rewrite\@S: will be used in each environment of nicematrix in order to "rewire" the S column in each environment.

\begin{verbatim}
\AtBeginDocument
\begin{verbatim}
183 \bool_if:nTF { ! \c_@@_siunitx\_loaded_bool }
184 { \cs_set_eq:NN \@@_renew\_\NC@rewrite\@S: \prg_do_nothing: }
185 { \cs_new_protected:Npn \@@_renew\_\NC@rewrite\@S: {
186 { \renewcommand*{\NC@rewrite}\@S\{}[\]
187 \\@temptokena \exp_after:wN
188 \\tex_the:D \\@temptokena
189 \} { \@@_Cell: \c_@@_table\_collect\_begin_tl S \{##1\} }
190 \@@_true_c: will be replaced statically by c at the end of the construction of the preamble.
191 \@@_true_c: < { \c_@@_table\_print_tl \@@_end_Cell: }
192 \NC@find
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
\end{verbatim}
\end{verbatim}

47
The following regex will be used to modify the preamble of the array when the key `colortbl-like` is used.

\regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

If the final user uses `nicematrix`, PGF/Tikz will write instruction \pgfsyspdfmark in the aux file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions \pgfsyspdfmark in the aux file. With the following code, we avoid that situation.

\cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
\iow_now:Nn \@mainaux
\ExplSyntaxOn
\cs_if_free:NT \pgfsyspdfmark
{ \cs_set_eq:NN \pgfsyspdfmark @gobblethree }
\ExplSyntaxOff
\cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
\}

Parameters

For compatibility with versions prior to 5.0, we provide a load-time option `define_L_C_R`. With this option, it’s possible the letters L, C and R instead of l, c and r in the preamble of the environments of `nicematrix` as it was mandatory before version 5.0.

\bool_new:N \c_@@_define_L_C_R_bool
\cs_new_protected:Npn \@@_define_L_C_R:
\newcolumntype L l
\newcolumntype C c
\newcolumntype R r
\}

The following counter will count the environments \{NiceArray\}. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

\int_new:N \g_@@_env_int

The following command is only a syntaxic shortcut. It must not be protected (it will be used in names of PGF nodes).

\cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }

The command \NiceMatrixLastEnv is not used by the package `nicematrix`. It’s only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it’s meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command must be expandable since it will be used in pgf nodes.

\NewExpandableDocumentCommand \NiceMatrixLastEnv { } { \int_use:N \g_@@_env_int }

The following command is only a syntaxic shortcut. The q in `qpoint` means quick.

\cs_new_protected:Npn \@@_qpoint:n #1
{ \pgfpointanchor { \@@_env: - #1 } { center } }

The following counter will count the environments \{NiceMatrixBlock\}.

\int_new:N \g_@@_NiceMatrixBlock_int
The dimension \_\_\_columns_width_dim will be used when the options specify that all the columns must have the same width (but, if the key columns-width is used with the special value auto, the boolean \_\_\_auto_columns_width_bool also will be raised).

\dim_new:N \_\_\_columns_width_dim

The following token list will contain the type of the current cell (l, c or r). It will be used by the blocks.

\tl_new:N \_\_\_cell_type_tl
\tl_set:Nn \_\_\_cell_type_tl { c }

When there is a mono-column block (created by the command \Block), we want to take into account the width of that block for the width of the column. That’s why we compute the width of that block in the \_\_\_blocks_width_dim and, after the construction of the box \_\_\_cell_box, we change the width of that box to take into account the length \_\_\_blocks_width_dim.

\dim_new:N \_\_\_blocks_width_dim

The sequence \_\_\_names_seq will be the list of all the names of environments used (via the option name) in the document: two environments must not have the same name. However, it’s possible to use the option allow-duplicate-names.

\seq_new:N \_\_\_names_seq

We want to know if we are in an environment of nicematrix because we will raise an error if the user tries to use nested environments.

\bool_new:N \_\_\_in_env_bool

If the user uses \{NiceArray\} or \{NiceTabular\} the flag \_\_\_NiceArray_bool will be raised.

\bool_new:N \_\_\_NiceArray_bool

If the user uses \{NiceTabular\} or \{NiceTabular*\}, we will raise the following flag.

\bool_new:N \_\_\_NiceTabular_bool

If the user uses \{NiceTabular*\}, the width of the tabular (in the first argument of the environment \{NiceTabular*\}) will be stored in the following dimension.

\dim_new:N \_\_\_tabular_width_dim

If the user uses an environment without preamble, we will raise the following flag.

\bool_new:N \_\_\_Matrix_bool

The following boolean will be raised when the command \rotate is used.

\bool_new:N \_\_\_rotate_bool

\cs_new_protected:Npn \_\_\_test_if_math_mode:
\{\if_mode_math: \else: \_\_\_fatal:n { Outside~math~mode } \fi: \}

The following colors will be used to memorize le color of the potential “first col” and the potential “first row”.

\colorlet { nicematrix-last-col } { . }
\colorlet { nicematrix-last-row } { . }
The following string is the name of the current environment or the current command of \nicematrix (despite its name which contains \env).
\str_new:N \g_@@_name_env_str

The following string will contain the word \command or \environment whether we are in a command of \nicematrix or in an environment of \nicematrix. The default value is \environment.
\tl_set:NN \g_@@_com_or_env_str \{ \environment \}

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains \env). This command must \not be protected since it will be used in error messages.
\cs_new:Npn \@@_full_name_env:
\{ \str_if_eq:VnnTF \g_@@_com_or_env_str \{ \command \} \{ \environment \} \}

The following token list corresponds to the key \rules/color available in the environments.
\tl_new:N \l_@@_rules_color_tl

A kind of false row will be inserted at the end of the array for the construction of the col nodes (and also to fix the width of the columns when \columns-width is used). When this special row will be created, we will raise the flag \g_@@_row_of_col_done_bool in order to avoid some actions set in the redefinition of \everycr when the last \cr of the \halign will occur (after that row of col nodes).
\bool_new:N \g_@@_row_of_col_done_bool

\l_@@_code_before_tl may contain two types of informations:

- A \code-before written in the aux file by a previous run. When the aux file is read, this \code-before is stored in \g_@@_code_before_i_tl (where \i is the number of the environment) and, at the beginning of the environment, it will be put in \l_@@_code_before_tl.

- The final user can explicitly add material in \l_@@_code_before_tl by using the key \code-before.

\bool_new:N \g_@@_row_of_col_done_bool
\tl_new:N \l_@@_code_before_tl
\bool_new:N \l_@@_code_before_bool
The following dimensions will be used when drawing the dotted lines.

\dim_new:N \l_@@_x_initial_dim
\dim_new:N \l_@@_y_initial_dim
\dim_new:N \l_@@_x_final_dim
\dim_new:N \l_@@_y_final_dim

 expl3 provides scratch dimension \l_tmpa_dim and \l_tmpd_dim. We create two others in the same spirit (if they don’t exist yet : that’s why we use \dim_zero_new:N).

\dim_zero_new:N \l_tmpc_dim
\dim_zero_new:N \l_tmpd_dim

Some cells will be declared as “empty” (for example a cell with an instruction \Cdots).

\bool_new:N \g_@@_empty_cell_bool

The following dimension will be used to save the current value of \arraycolsep.

\dim_new:N \@@_old_arraycolsep_dim

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

\dim_new:N \g_@@_width_last_col_dim
\dim_new:N \g_@@_width_first_col_dim

The following sequence will contain the characteristics of the blocks of the array, specified by the command \Block. Each block is represented by 6 components surrounded by braces: \{imin\}\{jmin\}\{imax\}\{jmax\}\{options\}\{contents\}.
The variable is global because it will be modified in the cells of the array.

\seq_new:N \g_@@_blocks_seq

We also manage a sequence of the positions of the blocks. Of course, it’s redundant with the previous sequence, but it’s for efficiency. In that sequence, each block is represented by only the four first components: \{imin\}\{jmin\}\{imax\}\{jmax\}.

\seq_new:N \g_@@_pos_of_blocks_seq

In fact, this sequence will also contain the positions of the cells with a \diagbox. The sequence \g_@@_pos_of_blocks_seq will be used when we will draw the rules (which respect the blocks).

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by \Cdots, \Vdots, \Ddots, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains four components: \{imin\}\{jmin\}\{imax\}\{jmax\}.

\seq_new:N \g_@@_pos_of_xdots_seq

The sequence \g_@@_pos_of_xdots_seq will be used when we will draw the rules required by the key hvlines (these rules won’t be drawn within the virtual blocks corresponding to the dotted lines).

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble, of course and without the potential exterior columns).

\int_new:N \g_@@_static_num_of_col_int

Used for the color of the blocks.

\tl_new:N \l_@@_color_tl

The parameter of position of the label of a block (c, r or l).

\tl_new:N \l_@@_pos_of_block_tl
\tl_set:Nn \l_@@_pos_of_block_tl { c }

Used when the key draw-first is used for \Ddots or \Iddots.

\bool_new:N \l_@@_draw_first_bool
The blocks which use the key \textarus{block} will store their content in a box. These boxes are numbered with the following counter.

\begin{verbatim}
\int_new:N \g_@@_block_box_int
\end{verbatim}

Variables for the exterior rows and columns

The keys for the exterior rows and columns are \textarus{first-row, first-col, last-row} and \textarus{last-col}. However, internally, these keys are not coded in a similar way.

- **First row**
  The integer \textarus{l_@@_first_row_int} is the number of the first row of the array. The default value is 1, but, if the option \textarus{first-row} is used, the value will be 0.

\begin{verbatim}
\int_new:N \l_@@_first_row_int
\int_set:Nn \l_@@_first_row_int 1
\end{verbatim}

- **First column**
  The integer \textarus{l_@@_first_col_int} is the number of the first column of the array. The default value is 1, but, if the option \textarus{first-col} is used, the value will be 0.

\begin{verbatim}
\int_new:N \l_@@_first_col_int
\int_set:Nn \l_@@_first_col_int 1
\end{verbatim}

- **Last row**
  The counter \textarus{l_@@_last_row_int} is the number of the potential “last row”, as specified by the key \textarus{last-row}. A value of \(\text{-2}\) means that there is no “last row”. A value of \(\text{-1}\) means that there is a “last row” but we don’t know the number of that row (the key \textarus{last-row} has been used without value and the actual value has not still been read in the aux file).

\begin{verbatim}
\int_new:N \l_@@_last_row_int
\int_set:Nn \l_@@_last_row_int \{ -2 \}
\end{verbatim}

If, in an environment like \textarus{pNiceArray}, the option \textarus{last-row} is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the aux file the number of the “last row”.

\begin{verbatim}
\bool_new:N \l_@@_last_row_without_value_bool
\end{verbatim}

Idem for \textarus{l_@@_last_col_without_value_bool}

\begin{verbatim}
\bool_new:N \l_@@_last_col_without_value_bool
\end{verbatim}

- **Last column**
  For the potential “last column”, we use an integer. A value of \(\text{-2}\) means that there is no last column. A value of \(\text{-1}\) means that we are in an environment without preamble (e.g. \textarus{bNiceMatrix}) and there is a last column but we don’t know its value because the user has used the option \textarus{last-col} without value. A value of 0 means that the option \textarus{last-col} has been used in an environment with preamble (like \textarus{pNiceArray}): in this case, the key was necessary without argument.

\begin{verbatim}
\int_new:N \l_@@_last_col_int
\int_set:Nn \l_@@_last_col_int \{ -2 \}
\end{verbatim}

\footnote{We can’t use \textarus{l_@@_last_row_int} for this usage because, if \textarus{nicematrix} has read its value from the aux file, the value of the counter won’t be \text{-1} any longer.}
However, we have also a boolean. Consider the following code:

```latex
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}
```

In such a code, the “last column” specified by the key `last-col` is not used. We want to be able to detect such a situation and we create a boolean for that job.

```latex
\bool_new:N \g_@@_last_col_found_bool
```

This boolean is set to `false` at the end of `@@_pre_array`.

### The command \tabularnote

The LaTeX counter `tabularnote` will be used to count the tabular notes during the construction of the array (this counter won’t be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```latex
\newcounter{tabularnote}
```

We will store in the following sequence the tabular notes of a given array.

```latex
\seq_new:N \g_@@_tabularnotes_seq
```

However, before the actual tabular notes, it’s possible to put a text specified by the key `tabularnote` of the environment. The token list `\l_@@_tabularnote_tl` corresponds to the value of that key.

```latex
\tl_new:N \l_@@_tabularnote_tl
```

The following counter will be used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. In the tabular, the labels of those nodes are composed as a comma separated list (e.g. `a,b,c`).

```latex
\int_new:N \l_@@_number_of_notes_int
```

The following function can be redefined by using the key `notes/style`.

```latex
\cs_new:Npn \@@_notes_style:n #1 { \textit{\alph{#1}} }
```

The following function can be redefined by using the key `notes/label-in-tabular`.

```latex
\cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript{#1} }
```

The following function can be redefined by using the key `notes/label-in-list`.

```latex
\cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript{#1} }
```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a footnote which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```latex
\cs_set:Npn \thetabularnote { \{ \@@_notes_style:n \{ \tabularnote \} \} }
```
The tabular notes will be available for the final user only when enumitem is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by enumitem (a list tabularnotes in the general case and a list tabularnotes* if the key para is in force). However, we can test whether enumitem has been loaded only at the beginning of the document (we want to allow the user to load enumitem after nicematrix).

\begin{Verbatim}
\AtBeginDocument{
\bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
{ \NewDocumentCommand \tabularnote { m }
{ \@@_error:n { enumitem-not-loaded } }
}
}
\begin{Verbatim}
The type of list tabularnotes will be used to format the tabular notes at the end of the array in the general case and tabularnotes* will be used if the key para is in force.

\begin{Verbatim}
\newlist { tabularnotes } { enumerate } { 1 }
\setlist [ tabularnotes ]
{ topsep = 0pt , noitemsep , leftmargin = * , align = left , labelsep = 0pt , label = \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } , }
\newlist { tabularnotes* } { enumerate* } { 1 }
\setlist [ tabularnotes* ]
{ afterlabel = \nobreak , itemjoin = \quad , label = \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } } , }
\end{Verbatim}

The command \tabularnote is available in the whole document (and not only in the environments of nicematrix) because we want it to be available in the caption of a \{table\} (before the following \{NiceTabular\} or \{NiceArray\}). That’s also the reason why the variables \c@tabularnote and \g_@@_tabularnotes_seq will be cleared at the end of the environment of nicematrix (and not at the beginning).

Unfortunately, if the package caption is loaded, the command \caption evaluates its argument twice and since it is not aware (of course) of \tabularnote, the command \tabularnote is, in fact, not usable in \caption when caption is loaded.\footnote{\textit{We should try to find a solution to that problem.}}

\begin{Verbatim}
\NewDocumentCommand \tabularnote { m }
{ \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
{ \@@_error:n { tabularnote-forbidden } }
\int_incr:N \l_@@_number_of_notes_int
\peek_meaning:NF \tabularnote
{ }
\end{Verbatim}
\begin{Verbatim}
\l_@@_number_of_notes_int is used to count the number of successive tabular notes such as in \tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}. We will have to compose the labels of these notes as a comma separated list (e.g. \textit{a,b,c}).

\begin{Verbatim}
\int_incr:N \l_@@_number_of_notes_int
\seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
\peek_meaning:NF \tabularnote
{ }
\end{Verbatim}
\end{Verbatim}
If the following token is not a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell.

```latex
\hbox_set:Nn \l_tmpa_box
{

We remind that it is the command `\@@_notes_label_in_tabular:n` that will (most of the time) put the labels in a `\textsuperscript`.

```latex
\@@_notes_label_in_tabular:n
{

\stepcounter { tabularnote }
\@@_notes_style:n { tabularnote }
\prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
{
\stepcounter { tabularnote }
\@@_notes_style:n { tabularnote }
}
}
```

We use `\refstepcounter` in order to have the (last) tabular note referenceable (with the standard command `\label`) and that’s why we have to go back with a decrementation of the counter `tabularnote` first.

```latex
\addtocounter { tabularnote } { -1 }
\refstepcounter { tabularnote }
\int_zero:N \l_@@_number_of_notes_int
\hbox_overlap_right:n { \box_use:N \l_tmpa_box }
```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```latex
\skip_horizontal:n { \box_wd:N \l_tmpa_box }
```

Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

#1 is the name of the node which will be created; #2 and #3 are the coordinates of one of the corner of the rectangle; #4 and #5 are the coordinates of the opposite corner.

```latex
\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
{
\begin { pgfscope }
\pgfset
{
outer~sep = \c_zero_dim ,
inner~sep = \c_zero_dim ,
minimum~size = \c_zero_dim
}
\pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
\pgfnode
{ rectangle }
{ center }
{ \vbox_to_ht:nn}
{ \dim_abs:n { #5 - #3 } }
```

55
The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgr_rect_node:nnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

\begin{verbatim}
\cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3 #4
{
\begin{pgfscope}
\pgfset{
outer sep = \c_zero_dim ,
inner sep = \c_zero_dim ,
minimum size = \c_zero_dim}
\pgftransformshift{\pgfpointscale{0.5}{\pgfpointadd{#2}{#3}}}
\pgfpointdiff{#3}{#2}
\pgfgetlastxy\l_tmpa_dim\l_tmpb_dim
\pgfnode{rectangle}{center}{\vbox_to_ht:nn{\dim_abs:n\l_tmpb_dim}{\vfill\hbox_to_wd:nn{\dim_abs:n\l_tmpa_dim}{}}}{#1}{}}
\end{pgfscope}
\end{verbatim}

The options

By default, the commands `\cellcolor` and `\rowcolor` are available for the user in the cells of the tabular (the user may use the commands provided by `colortbl`). However, if the key `colortbl-like` is used, these commands are available.

\begin{verbatim}
\bool_new:N \l_@@_colortbl_like_bool
\end{verbatim}

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidht`. It’s possible to disable this feature with the key `\l_@@_standard_line_bool`.

\begin{verbatim}
\bool_new:N \l_@@_standard_cline_bool
\end{verbatim}

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

\begin{verbatim}
\dim_new:N \l_@@_cell_space_top_limit_dim
\dim_new:N \l_@@_cell_space_bottom_limit_dim
\end{verbatim}

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

\begin{verbatim}
\dim_new:N \l_@@_inter_dots_dim
\dim_set:Nn \l_@@_inter_dots_dim {0.45 \em}
\end{verbatim}

56
The following dimension is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).

\dim_new:N \l_@@_xdots_shorten_dim
\dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 \em }

The following dimension is the radius of the dots for the dotted lines (when line-style is equal to standard, which is the initial value). The initial value is 0.53 pt but it will be changed if the option small is used.

\dim_new:N \l_@@_radius_dim
\dim_set:Nn \l_@@_radius_dim { 0.53 \pt }

The token list \l_@@_xdots_line_style_tl corresponds to the option tikz of the commands \Cdots, \Ldots, etc. and of the options line-style for the environments and \NiceMatrixOptions. The constant \c_@@_standard_tl will be used in some tests.

\tl_new:N \l_@@_xdots_line_style_tl
\tl_const:Nn \c_@@_standard_tl { standard }
\tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

The boolean \l_@@_light_syntax_bool corresponds to the option light-syntax.

\bool_new:N \l_@@_light_syntax_bool

The string \l_@@_baseline_str may contain one of the three values t, c or b as in the option of the environment \{array\}. However, it may also contain an integer (which represents the number of the row to which align the array).

\str_new:N \l_@@_baseline_str
\tl_set:Nn \l_@@_baseline_str c

The flag \l_@@_exterior_arraycolsep_bool corresponds to the option exterior-arraycolsep. If this option is set, a space equal to \arraycolsep will be put on both sides of an environment \{NiceArray\} (as it is done in \{array\} of array).

\bool_new:N \l_@@_exterior_arraycolsep_bool

The flag \l_@@_parallelize_diags_bool controls whether the diagonals are parallelized. The initial value is true.

\bool_new:N \l_@@_parallelize_diags_bool
\bool_set_true:N \l_@@_parallelize_diags_bool

If the flag \l_@@_vlines_bool is raised, horizontal space will be reserved in the the preamble of the array (for the vertical rules) and, after the construction of the array, the vertical rules will be drawn.

\bool_new:N \l_@@_vlines_bool

If the flag \l_@@_hlines_bool is raised, vertical space will be reserved between the rows of the array (for the horizontal rules) and, after the construction of the array, the vertical rules will be drawn.

\bool_new:N \l_@@_hlines_bool

The flag \l_@@_except_corners_bool will be raised when the key except-corners will be used. In that case, the corners will be computed before we draw rules and the rules won’t be drawn in the corners. As expected, the key hlines-except-corners raises the key except-corners.

\clist_new:N \l_@@_except_corners_clist
\dim_new:N \l_@@_notes_above_space_dim
\dim_set:Nn \l_@@_notes_above_space_dim { 1 \mm }
The flag \l_@@_nullify_dots_bool corresponds to the option nullify-dots. When the flag is
down, the instructions like \vdots are inserted within a \hphantom (and so the constructed matrix
has exactly the same size as a matrix constructed with the classical \{matrix\} and \ldots, \vdots,
etc.).

\bool_new:N \l_@@_nullify_dots_bool

The following flag will be used when the current options specify that all the columns of the array
must have the same width equal to the largest width of a cell of the array (except the cells of the
potential exterior columns).

\bool_new:N \l_@@_auto_columns_width_bool

The string \l_@@_name_str will contain the optional name of the environment: this name can be
used to access to the Tikz nodes created in the array from outside the environment.

\str_new:N \l_@@_name_str

The boolean \l_@@_medium_nodes_bool will be used to indicate whether the “medium nodes” are
created in the array. Idem for the “large nodes”.

\bool_new:N \l_@@_medium_nodes_bool
\bool_new:N \l_@@_large_nodes_bool

The dimension \l_@@_left_margin_dim correspond to the option left-margin. Idem for the right
margin. These parameters are involved in the creation of the “medium nodes” but also in the
placement of the delimiters and the drawing of the horizontal dotted lines (\hdottedline).

\dim_new:N \l_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim

The dimensions \l_@@_extra_left_margin_dim and \l_@@_extra_right_margin_dim correspond to the options extra-left-margin and extra-right-margin.

\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim

The token list \l_@@_end_of_row_tl corresponds to the option end-of-row. It specifies the symbol
used to mark the ends of rows when the light syntax is used.

\tl_new:N \l_@@_end_of_row_tl
\tl_set:Nn \l_@@_end_of_row_tl { ; }

The following parameter is for the color the dotted lines drawn by \Cdots, \Ldots, \Vdots, \Ddots,
\Iddots and \Hdots for but not the dotted lines drawn by \hdottedline and “:”.

\tl_new:N \l_@@_xdots_color_tl

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the
columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns
(for example with the option columns-width or the option auto-columns-width of the environment
\{NiceMatrixBlock\}). However, even if we use the same type of delimiters, the width of the delimiters
may be different from an array to another because the width of the delimiter is function of its size.
That’s why we create an option called max-delimiter-width which will give to the delimiters the
width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

\bool_new:N \l_@@_max_delimiter_width_bool

\keys_define:nn { NiceMatrix / xdots } { line-style .code:n = \bool_lazy_or:nnTF }
We can't use _tikz_loaded_bool_ to test whether tikz is loaded because _NiceMatrixOptions_ may be used in the preamble of the document.

```latex
\ifcs_if_exist_p:N \tikzpicture \\
\ifstr_if_eq_p:nn { #1 } { standard } \\
\tl_set:Nn \l_@@_xdots_line_style_tl { #1 } \\
\@@_error:n { \texttt{bad\text{-}option\text{-}for\text{-}line\text{-}style} } \\
\end{verbatim}
\texttt{line-style .value_required:n = true} ,
\texttt{color .tl_set:N = \l_@@_xdots_color_tl} ,
\texttt{color .value_required:n = true} ,
\texttt{shorten .dim_set:N = \l_@@_xdots_shorten_dim} ,
\texttt{shorten .value_required:n = true} ,
```

The options down and up are not documented for the final user because he should use the syntax with `^` and `_`.

```latex
\down .tl_set:N = \l_@@_xdots_down_tl ,
\up .tl_set:N = \l_@@_xdots_up_tl ,
```

The key draw-first, which is meant to be used only with \Ddots and \Iddots, which be catched when \Ddots or \Iddots is used (during the construction of the array and not when we draw the dotted lines).

```latex
draw-first .code:n = \prg_do_nothing: ,
```

```latex
\keys_define:nn { NiceMatrix / rules } 
\{ 
\color .tl_set:N = \l_@@_rules_color_tl ,
\color .value_required:n = true ,
\width .dim_set:N = \arrayrulewidth ,
\width .value_required:n = true 
\}
```

First, we define a set of keys “NiceMatrix / Global” which will be used (with the mechanism of \_\_inherit:n) by other sets of keys.

```latex
\keys_define:nn { NiceMatrix / Global } 
\{ 
\standard_cline .bool_set:N = \l_@@_standard_cline_bool ,
\standard_cline .default:n = true ,
\cell_space_top_limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
\cell_space_top_limit .value_required:n = true ,
\cell_space_bottom_limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
\cell_space_bottom_limit .value_required:n = true ,
\xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
\maxdelimiter_width .bool_set:N = \l_@@_max_delimiter_width_bool ,
\light_syntax .bool_set:N = \l_@@_light_syntax_bool ,
\light_syntax .default:n = true ,
\end_of_row .value_required:n = true ,
\first_col .code:n = \int_zero:N \l_@@_first_col_int ,
\first_row .code:n = \int_zero:N \l_@@_first_row_int ,
\last_row .int_set:N = \l_@@_last_row_int ,
\last_row .default:n = -1 ,
\code_for_first_col .tl_set:N = \l_@@_code_for_first_col_tl ,
\code_for_first_col .value_required:n = true ,
\code_for_last_col .tl_set:N = \l_@@_code_for_last_col_tl ,
\code_for_last_col .value_required:n = true ,
\code_for_first_row .tl_set:N = \l_@@_code_for_first_row_tl ,
\code_for_first_row .value_required:n = true ,
\code_for_last_row .tl_set:N = \l_@@_code_for_last_row_tl ,
\code_for_last_row .value_required:n = true ,
\hlines .bool_set:N = \l_@@_hlines_bool ,
\}
```
With the option `renew-dots`, the command \cdots, \ldots, \vdots, \ddots, etc. are redefined and behave like the commands \Cdots, \Ldots, \Vdots, \Ddots, etc.

\keys_define:nn { NiceMatrix / Env }
{
  except-corners .clist_set:N = \l_@@_except_corners_clist ,
  except-corners .default:n = \{ NW, SW, NE, SE \} ,
  hvlines-except-corners .code:n =
  \clist_set:Nn \l_@@_except_corners_clist { #1 }
  \bool_set_true:N \l_@@_vlines_bool
  \bool_set_true:N \l_@@_hlines_bool
},

The options c, t and b of the environment {NiceArray} have the same meaning as the option of the classical environment {array}.

c .code:n = \tl_set:Nn \l_@@_baseline_str c ,
t .code:n = \tl_set:Nn \l_@@_baseline_str t ,
b .code:n = \tl_set:Nn \l_@@_baseline_str b ,
baseline .tl_set:N = \l_@@_baseline_str ,
baseline .value_required:n = true ,
columns-width .code:n =
\tl_if_eq:nnTF { #1 } \{ auto \}
We test whether we are in the measuring phase of an environment of amsmath (always loaded by nicematrix) because we want to avoid a fallacious message of duplicate name in this case.

\legacy_if:nF { measuring0 }
{
\str_set:Nn \l_tmpa_str { #1 }
\seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
{ \@@_error:nn { Duplicate~name } { #1 } }
{ \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
\str_set_eq:NN \l_@@_name_str \l_tmpa_str
}

\keys_define:nn { NiceMatrix / notes }
{
\bool_set:N \l_@@_notes_para_bool,
\bool_set:N \l_@@_notes_bottomrule_bool,
\bool_set:N \l_@@_colortbl_like_bool,
\bool_set:N \l_@@_code_before Bool,
\bool_set_true:N \l_@@_auto_columns_widthBool
\dim_set:Nn \l_@@_columns_width_dim { #1 }
\bool_set_true:N \l_@@_auto_columns_width_bool
columns-width .value_required:n = true,
name .code:n =
\legacy_if:nF { measuring0 }
{
\str_set:Nn \l_tmpa_str { #1 }
\seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
{ \@@_error:nn { Duplicate~name } { #1 } }
{ \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
\str_set_eq:NN \l_@@_name_str \l_tmpa_str
},
name .value_required:n = true,
code-after .value_required:n = true,
colortbl-like .code:n =
\bool_set_true:N \l_@@_colortbl_like_bool,
\bool_set_true:N \l_@@_code_before_bool,
colortbl-like .value_forbidden:n = true
}
\keys_define:nn { NiceMatrix / notes }

\bool_if:NTF \c_@@_in_preamble_bool
{
\AtBeginDocument
{
\bool_if:NT \c_@@_enumitem_loaded_bool
{ \setlist* [ tabularnotes ] { #1 } }
}
}
\bool_if:NTF \c_@@_in_preamble_bool
{ \setlist* [ tabularnotes ] { #1 } }
\enumitem-keys .value_required:n = true,
\enumitem-keys-para .code:n =
\bool_if:NTF \c_@@_in_preamble_bool
{
\AtBeginDocument
{
\bool_if:NT \c_@@_enumitem_loaded_bool
{ \setlist* [ tabularnotes ] { #1 } }
}
\bool_if:NT \c_@@_enumitem_loaded_bool
{ \setlist* [ tabularnotes ] { #1 } }
\enumitem-keys .value_required:n = true,
\enumitem-keys-para .code:n =
\bool_if:NTF \c_@@_in_preamble_bool
{
\AtBeginDocument
{
\bool_if:NT \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
}

\bool_if:NTF \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
\enumitem-keys .value_required:n = true,
\enumitem-keys-para .code:n =
\bool_if:NTF \c_@@_in_preamble_bool
{
\AtBeginDocument
{
\bool_if:NT \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
}

\bool_if:NTF \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
\enumitem-keys .value_required:n = true,
\enumitem-keys-para .code:n =
\bool_if:NTF \c_@@_in_preamble_bool
{
\AtBeginDocument
{
\bool_if:NT \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
}

\bool_if:NTF \c_@@_enumitem_loaded_bool
{ 
\setlist* [ tabularnotes ] { #1 } }
\enumitem-keys .value_required:n = true,
We begin the construction of the major sets of keys (used by the different user commands and environments).

\begin{verbatim}
\keys_define:nn { NiceMatrix }
\begin{Verbatim}
\texttt{NiceMatrixOptions .inherit:n = }
\begin{Verbatim}
\texttt{\{ NiceMatrix / Global } ,
\texttt{ NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,}
\texttt{ NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,}
\texttt{ NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,}
\texttt{ NiceMatrix .inherit:n = }
\begin{Verbatim}
\texttt{\{ NiceMatrix / Global ,}
\texttt{ NiceMatrix / Env ,}
\end{Verbatim}
\begin{Verbatim}
\texttt{\} ,}
\texttt{ NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,}
\texttt{ NiceMatrix / rules .inherit:n = NiceMatrix / rules ,}
\texttt{ NiceTabular .inherit:n = }
\begin{Verbatim}
\texttt{\{ NiceMatrix / Global ,}
\texttt{ NiceMatrix / Env ,}
\end{Verbatim}
\begin{Verbatim}
\texttt{\} ,}
\texttt{ NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,}
\texttt{ NiceTabular / rules .inherit:n = NiceMatrix / rules ,}
\texttt{ NiceArray .inherit:n = }
\begin{Verbatim}
\texttt{\{ NiceMatrix / Global ,}
\texttt{ NiceMatrix / Env ,}
\end{Verbatim}
\begin{Verbatim}
\texttt{\} ,}
\texttt{ NiceArray / xdots .inherit:n = NiceMatrix / xdots ,}
\texttt{ NiceArray / rules .inherit:n = NiceMatrix / rules ,}
\texttt{ pNiceArray .inherit:n = }
\begin{Verbatim}
\texttt{\{ NiceMatrix / Global ,}
\texttt{ NiceMatrix / Env ,}
\end{Verbatim}
\begin{Verbatim}
\texttt{\} ,}
\texttt{ pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,}
\texttt{ pNiceArray / rules .inherit:n = NiceMatrix / rules ,}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{verbatim}

We finalise the definition of the set of keys “NiceMatrix / NiceMatrixOptions” with the options specific to NiceMatrixOptions.

\begin{verbatim}
\keys_define:nn { NiceMatrix / NiceMatrixOptions }
\begin{Verbatim}
\texttt{last-col .code:n = \tl_if_empty:nF { #1 }
\begin{Verbatim}
\texttt{\{ \@_error:n \{ last-col-non-empty-for-NiceMatrixOptions \} }
\begin{Verbatim}
\texttt{\int_zero:N \l_@@_last_col_int ,}
\texttt{ small .bool_set:N = \l_@@_small_bool ,}
\texttt{ small .value_forbidden:n = true ,}
\end{Verbatim}
\begin{Verbatim}
\texttt{\}}
\end{Verbatim}
\end{Verbatim}
\end{verbatim}

With the option renew-matrix, the environment \texttt{\{matrix\}} of amsmath and its variants are redefined to behave like the environment \texttt{\{NiceMatrix\}} and its variants.

62
The key `transparent` is now considered as obsolete (because its name is ambiguous).

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

If the option `columns-width` is used, all the columns will have the same width.

In `{NiceMatrixOptions}`, the special value `auto` is not available.

By default, the specifier used in the preamble of the array (for example in `{pNiceArray}`) to draw a vertical dotted line between two columns is the colon `:`. However, it’s possible to change this letter with `letter-for-dotted-lines` and, by the way, the letter `:` will remain free for other packages (for example `arydshln`).

We finalise the definition of the set of keys “{NiceMatrix / NiceMatrix}” with the options specific to `{NiceMatrix}`.
We finalise the definition of the set of keys “NiceMatrix / NiceArray” with the options specific to \{NiceArray\}.

\keys_define:nn { NiceMatrix / NiceArray }
{
  \keys_define:nn { NiceMatrix / pNiceArray }
  {
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    last-col .code:n = \tl_if_empty:nF {#1}
      \@@_error:n { last-col~non~empty~for~NiceArray } }

    \keys_define:nn { NiceMatrix / NiceTabular }
  {
    notes / para .bool_set:N = \l_@@_notes_para_bool ,
    notes / para .default:n = true ,
    notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
    notes / bottomrule .default:n = true ,
    tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
    tabularnote .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular } }
}

We finalise the definition of the set of keys “NiceMatrix / NiceTabular” with the options specific to \{NiceTabular\}.
Important code used by \{NiceArrayWithDelims\}

The pseudo-environment \@@_Cell:–\@@_end_Cell: will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a \halign (via an environment \{array\}).

\cs_new_protected:Npn \@@_Cell:
{\CodeAfter\omit\CodeAfter\i:n}

We increment \c@jCol, which is the counter of the columns.

\int_gincr:N \c@jCol

Now, we increment the counter of the rows. We don’t want to take into account because some packages, like \texttt{arydshln}, create special rows in the \halign that we don’t want to take into account.

\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
\int_gincr:N \c@jCol
\int_compare:nNnT \c@iRow = \l_@@_first_col_int \@@_begin_of_row: 
\l_@@_code_for_first_row_tl
\xglobal \colorlet { nicematrix-first-row } { . }
\int_compare:nNnTF \c@iRow = 0 
\l_@@_code_for_last_row_tl
\xglobal \colorlet { nicematrix-last-row } { . }
\int_compare:nNnTF \c@iRow = \l_@@_last_row_int
\l_@@_code_for_last_row_tl
\xglobal \colorlet { nicematrix-last-row } { . }

The following macro \@@_begin_of_row: is usually used in the cell number 1 of the row. However, when the key first-col is used, \@@_begin_of_row: is executed in the cell number 0 of the row.

\cs_new_protected:Npn \@@_begin_of_row:
{\dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
\dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
\pgfpicture
\pgfresetanchor{picturepositionnonpagetrue}
\pgfcoordinate
}
The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```latex
\cs_new_protected:Npn \@@_update_for_first_and_last_row: 
{ \int_compare:nNnTF \c@iRow = 0 
{ \dim_gset:Nn \g_@@_dp_row_zero_dim { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } } 
\dim_gset:Nn \g_@@_ht_row_zero_dim { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } } } 
{ \int_compare:nNnT \c@iRow = 1 
{ \dim_gset:Nn \g_@@_ht_row_one_dim { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } } } } 
} 
\cs_new_protected:Npn \@@_rotate_cell_box: 
{ \box_rotate:Nn \l_@@_cell_box { 90 } \int_compare:nNnT \c@iRow = \l_@@_last_row_int 
{ \vbox_set_top:Nn \l_@@_cell_box } 
\vbox_to_zero:n { \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 \textex } -- } 
\box_use:N \l_@@_cell_box } 
\bool_gset_false:N \g_@@_rotate_bool 
\cs_new_protected:Npn \@@_adjust_width_box: 
{ \dim_compare:nNnT \g_@@_blocks_width_dim > \c_zero_dim 
{ \box_set_wd:Nn \l_@@_cell_box } 
\dim_gset:Nn \g_@@_blocks_width_dim { \dim_max:nn \box_wd:N \l_@@_cell_box } 
\dim_gzero:N \g_@@_blocks_width_dim } 
\cs_new_protected:Npn \@@_end_Cell: 
{ \@@_math_toggle_token: \hbox_set_end: \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box: \@@_adjust_width_box: 
}
We want to compute in \texttt{\g_@@_max_cell_width_dim} the width of the widest cell of the array (except the cells of the “first column” and the “last column”).
\begin{verbatim}
\dim_gset:Nn \g_@@_max_cell_width_dim 
{ \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
\end{verbatim}

The following computations are for the “first row” and the “last row”.
\begin{verbatim}
@@_update_for_first_and_last_row:
\end{verbatim}

If the cell is empty, or may be considered as if, we must not create the \texttt{PGF} node, for two reasons:
- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s very difficult to determine whether a cell is empty. As of now, we use the following technic:
- if the width of the box \texttt{\l_@@_cell_box} (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a \texttt{\rlap}, a \texttt{\llap} or a \texttt{\mathclap} of \texttt{mathtools}.
- the cells with a command \texttt{\Ldots} or \texttt{\Cdots}, \texttt{\Vdots}, etc., should also be considered as empty: if \texttt{nullify-dots} is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of \texttt{code-after}); however, if \texttt{nullify-dots} is not in force, a phantom of \texttt{\ldots}, \texttt{\cdots}, \texttt{\vdots} is inserted and its width is not equal to zero; that’s why these commands raise a boolean \texttt{\g_@@_empty_cell_bool} and we begin by testing this boolean.

\begin{verbatim}
\bool_if:NTF \g_@@_empty_cell_bool 
{ \box_use_drop:N \l_@@_cell_box }
\end{verbatim}

The following command creates the \texttt{PGF} name of the node with, of course, \texttt{\l_@@_cell_box} as the content.
\begin{verbatim}
\cs_new_protected:Npn \@@_node_for_the_cell:
\end{verbatim}
The second argument of the following command \texttt{\@@\_instruction\_of\_type:nnn} defined below is the type of the instruction (Cdots, Vdots, Ddots, etc.). The third argument is the list of options. This command writes in the corresponding \texttt{\g_{@@}\_type\_lines\_tl} the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,
\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots \textcolor{red}{[color=red]} \\
\end{pNiceMatrix}
\begin{align*}
\begin{pmatrix}
1 & 2 & 3 & 4 \\
5 & & & 6 \\
7 & & & \\
\end{pmatrix}
\end{align*}
the content of \texttt{\g_{@@}\_Cdots\_lines\_tl} will be:
\begin{enumerate}
\item \texttt{\@@\_draw\_Cdots:nnn {2}{2}{}}
\item \texttt{\@@\_draw\_Cdots:nnn {3}{2}{color=red}}
\end{enumerate}

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions.

It’s important to use a \texttt{\tl_gput\_right:cx} and not a \texttt{\tl_gput\_left:cx} because we want the \texttt{\Ddots} lines to be drawn in the order of appearance in the array (for parallelisation).

We want to use \texttt{\array} of \texttt{array}. However, if the class used is \texttt{revtex4-1} or \texttt{revtex4-2}, we have to do some tuning and use the command \texttt{\@array@array} instead of \texttt{\array} because these classes do a redefinition of \texttt{\array} incompatible with our use of \texttt{\array}.
It \texttt{colortbl} is loaded, \texttt{@tabarray} has been redefined to incorporate \texttt{\CT@start}.

\begin{verbatim}
\@tabarray
\l_{\texttt{\@baseline_str}} may have the value $t$, $c$ or $b$. However, if the value is $b$, we compose the \texttt{array} 
(of array) with the option $t$ and the right translation will be done further.

\begin{verbatim}
[ \str_if_eq:VnTF \l_{\texttt{\@baseline_str}} c c t ]
\end{verbatim}

We keep in memory the standard version of \texttt{\ialign} because we will redefine \texttt{\ialign} in the environment \{NiceArrayWithDelims\} but restore the standard version for use in the cells of the array.

\begin{verbatim}
\cs_set_eq:NN \@@_old_ialign: \ialign
\end{verbatim}

The following command creates a row node (and not a row of nodes!).

\begin{verbatim}
\cs_new_protected:Npn \@@_create_row_node:
\end{verbatim}

\begin{verbatim}
\hbox:n (or \hbox) is mandatory.
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i:
\end{verbatim}
\end{verbatim}

We don’t draw the rules of the key \texttt{hlines} (or \texttt{hvlines}) but we reserve the vertical space for these rules.

\begin{verbatim}
\bool_if:NT \l_{\texttt{\@hlines_bool}}
\end{verbatim}

The counter \texttt{\@iRow} has the value $-1$ only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

\begin{verbatim}
\int_compare:nNnT \c@iRow > \{-1\}
\end{verbatim}

\begin{verbatim}
\int_compare:nNnF \c@iRow = \l_{\texttt{\@last_row_int}}
\end{verbatim}

69
The command \CT@arc@ is a command of \texttt{colortbl} which sets the color of the rules in the array. The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded. We use a \TeX{} group in order to limit the scope of \CT@arc@.

\begin{verbatim}
{ \hrule height \arrayrulewidth width \c_zero_dim }
\end{verbatim}

The command \@@_newcolumntype is the command \texttt{\newcolumntype} of \texttt{array} without the warnings for redefinitions of columns types (we will use it to redefine the columns types \texttt{w} and \texttt{w}).

\begin{verbatim}
\cs_set_protected:Npn \@@_newcolumntype #1
\{
 \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
 \peek_meaning:NTF \[
 \{ \newcol@ #1 \}
 \{ \newcol@ #1 [ 0 ] \}
\}
\end{verbatim}

When the key \texttt{renew-dots} is used, the following code will be executed.

\begin{verbatim}
\cs_set_protected:Npn \@@_renew_dots:
\{
 \cs_set_eq:NN \ldots \@@_Ldots
 \cs_set_eq:NN \cdots \@@_Cdots
 \cs_set_eq:NN \vdots \@@_Vdots
 \cs_set_eq:NN \ddots \@@_Ddots
 \cs_set_eq:NN \iddots \@@_Iddots
 \cs_set_eq:NN \dots \@@_Ldots
 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
\}
\end{verbatim}

When the key \texttt{colortbl-like} is used, the following code will be executed.

\begin{verbatim}
\cs_new_protected:Npn \@@_colortbl_like:
\{
 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
 \cs_set_eq:NN \columncolor \@@_columncolor_preamble
\}
\end{verbatim}

The following code \texttt{\@@_pre_array:} is used in \texttt{\texttt{\{NiceArrayWithDelims\}}}\. It exists as a standalone macro only for legibility.

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_array:
\{
\end{verbatim}

If \texttt{booktabs} is loaded, we have to patch the macro \texttt{\BTnormal} which is a macro of \texttt{booktabs}. The macro \texttt{\BTnormal} draws an horizontal rule but it occurs after a vertical skip done by a low level \TeX{} command. When this macro \texttt{\BTnormal} occurs, the \texttt{row} node has yet been inserted by \texttt{nicematrix} before the vertical skip (and thus, at a wrong place). That why we decide to create a new \texttt{row} node (for the same row). We patch the macro \texttt{\BTnormal} to create this \texttt{row} node. This new \texttt{row} node will overwrite the previous definition of that \texttt{row} node and we have managed to avoid the error messages of that redefinition.\footnote{cf. \texttt{nicematrix@redefine@check@rerun}}
If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```latex
\bool_if:NT \l_@@_small_bool {
    \cs_set_nopar:Npn \arraystretch { 0.47 }
    \dim_set:Nn \arraycolsep { 1.45 \text{ pt} }
}
```

The environment `{array}` uses internally the command `{\ialign}`. We change the definition of `{\ialign}` for several reasons. In particular, `{\ialign}` sets `{\everycr}` to `{ }` and we need to have to change the value of `{\everycr}`.

```latex
\cs_set_nopar:Npn \ialign {
    \bool_if:NTF \c_@@_colortbl_loaded_bool {
        \CT@everycr {
            \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
            \@@_everycr: }
        } {
            \everycr { \@@_everycr: }
        }
    \tabskip = \c_zero_skip
}
```

The box `{\@arstrutbox}` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current values of `{\arraystretch}` and `{\extrarowheight}` of `{array}`. That box is inserted (via `{\@arstrut}`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `{\@arstrutbox}` and that's why we do it in the `{\ialign}`.

```latex
\dim_gzero_new:N \g_@@_dp_row_zero_dim
\dim_gzero_new:N \g_@@_ht_row_zero_dim
\dim_gzero_new:N \g_@@_ht_row_one_dim
\dim_gzero_new:N \g_@@_dp_last_row_dim
\dim_gzero_new:N \g_@@_ht_last_row_dim
\dim_gzero_new:N \g_@@_ht_one_dim
\dim_gzero_new:N \g_@@_dp_nte_last_row_dim
```

After its first use, the definition of `{\ialign}` will revert automatically to its default definition. With this programmation, we will have, in the cells of the array, a clean version of `{\ialign}`.

```latex
\cs_set_eq:NN \ialign \@@_old_ialign:
\halign
```

We keep in memory the old versions or `{\ldots}`, `{\cdots}`, etc. only because we use them inside `{\phantom}` commands in order that the new commands `{\ldots}`, `{\cdots}`, etc. give the same spacing (except when the option `nullify-dots` is used).

```latex
\cs_set_eq:NN \@@_old_ldots \ldots
\cs_set_eq:NN \@@_old_cdots \cdots
\cs_set_eq:NN \@@_old_vdots \vdots
\cs_set_eq:NN \@@_old_ddots \ddots
```

The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.
The sequence $\g_@@_multicolumn_cells_seq$ will contain the list of the cells of the array where a command $\multicolumn{n}{...}{...}$ with $n > 1$ is issued. In $\g_@@_multicolumn_sizes_seq$, the “sizes” (that is to say the values of $n$) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

The counter $\c@iRow$ will be used to count the rows of the array (its incrementation will be in the first cell of the row).

At the end of the environment \{array\}, $\c@iRow$ will be the total number of rows. $\g_@@_row_total_int$ will be the number of rows excepted the last row (if $\l_@@_last_row_bool$ has been raised with the option last-row).

The counter $\c@jCol$ will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter $\g_@@_col_total_int$. These counters are updated in the command \@@_Cell: executed at the beginning of each cell.

During the construction of the array, the instructions \Cdots, \Ldots, etc. will be written in token lists $\g_@@_Cdots_lines_tl$, etc. which will be executed after the construction of the array.

This is the end of $\@@_pre_array$:.
The environment \{NiceArrayWithDelims\}

\NewDocumentEnvironment{NiceArrayWithDelims}{mm}{m !}{m}

The aim of the following \texttt{group} (the corresponding \texttt{egroup} is, of course, at the end of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

\begin{group}
\tl_set:Nn \l_@@_left_delim_tl { #1 }
\tl_set:Nn \l_@@_right_delim_tl { #2 }
\int_gzero:N \g_@@_block_box_int
\dim_zero:N \g_@@_width_last_col_dim
\dim_zero:N \g_@@_width_first_col_dim
\bool_gset_false:N \g_@@_row_of_col_done_bool
\str_if_empty:NT \g_@@_name_env_str
  \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims }
\@@_adapt_S_column:
\bool_if:NTF \l_@@_NiceTabular_bool
  \mode_leave_vertical:
\@@_test_if_math_mode:
\bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
\bool_set_true:N \l_@@_in_env_bool
\cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@
\cs_if_exist:NT \tikz@library@external@loaded
  \tikzexternaldisable
\cs_if_exist:NT \ifstandalone
  \tikzset{external/optimize=false}
\end{group}

We deactivate Tikz externalization because we will use PGF pictures with the options overlay and remember picture (or equivalent forms). We deactivate with \texttt{tikzexternaldisable} and not with \texttt{tikzset{external/export=false}} which is not equivalent.

We increment the counter \texttt{g_@@_env_int} which counts the environments of the package.

\begin{verbatim}
\int_gincr:N \g_@@_env_int
\bool_if:NT \l_@@_code_before_bool
  \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
    \int_use:N \g_@@_env_int _ tl
\end{verbatim}

The sequence \texttt{g_@@_blocks_seq} will contain the carateristics of the blocks (specified by \texttt{Block}) of the array. The sequence \texttt{g_@@_pos_of_blocks_seq} will contain only the position of the blocks. Of course, this is redundant but it’s for efficiency.

\begin{verbatim}
\seq_gclear:N \g_@@_blocks_seq
\seq_gclear:N \g_@@_pos_of_blocks_seq
\seq_gclear:N \g_@@_pos_of_xdots_seq
\tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
  \bool_set_true:N \l_@@_code_before_bool
  \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
    \int_use:N \g_@@_env_int _ tl
\end{verbatim}

In fact, the sequence \texttt{g_@@_pos_of_blocks_seq} will also contain the positions of the cells with a \texttt{diagbox}.

\begin{verbatim}
\tl_if_exist:cT { g_@@_code_before_ \int_use:N \g_@@_env_int _ tl }
  \bool_set_true:N \l_@@_code_before_bool
  \exp_args:NNv \tl_put_right:Nn \l_@@_code_before_tl
    \int_use:N \g_@@_env_int _ tl
\end{verbatim}

The set of keys is not exactly the same for \{NiceArray\} and for the variants of \{NiceArray\} (\{pNiceArray\}, \{bNiceArray\}, etc.) because, for \{NiceArray\}, we have the options \texttt{t}, \texttt{c}, \texttt{b} and \texttt{baseline}.

\footnote{e.g. \texttt{\color{rgb}{0.5,0.5,0}}}

36
If the key `code-before` is used, we have to create the `col` nodes and the `row` nodes before the creation of the array. First, we have to test whether the size of the array has been written in the `aux` file in a previous run. In this case, a command `@@_size_nb_of_env:` has been created.

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `code-before` (and in the `code-after`) they represent the numbers of rows and columns of the array (without the potential last row and last column).

We have to adjust the values of `\c@iRow` and `\c@jCol` to take into account the potential last row and last column. A value of \( -2 \) for `\l_@@_last_row_int` means that there is no last row. Idem for the columns.

Now, we will create all the `col` nodes and `row` nodes with the informations written in the `aux` file. You use the technique described in the page 1229 of `pgfmanual.pdf`, version 3.1.4b.

First, the creation of the `row` nodes.

Now, the creation of the `col` nodes.
We compose the code-before in math mode in order to nullify the spaces put by the user between instructions in the code-before.

A value of $-1$ for the counter $\l_@@_last_row_int$ means that the user has used the option last-row without value, that is to say without specifying the number of that last row. In this case, we try to read that value from the aux file (if it has been written on a previous run).

A value based on the name is more reliable than a value based on the number of the environment.

A value of $-1$ for the counter $\l_@@_last_col_int$ means that the user has used the option last-col without value, that is to say without specifying the number of that last column. In this case, we try to read that value from the aux file (if it has been written on a previous run).
The code in \@@_pre_array: is used only by \{NiceArrayWithDelims\}.

\@@_pre_array:

We compute the width of the two delimiters.
\dim_zero_new:N \l_@@_left_delim_dim
\dim_zero_new:N \l_@@_right_delim_dim
\bool_if:NTF \l_@@_NiceArray_bool
{ \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep } }
\dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
\dim_zero_new:N \l_@@_left_delim_dim
\dim_zero_new:N \l_@@_right_delim_dim
\bool_if:NTF \l_@@_light_syntax_bool
{ \use:c \{ @@-light-syntax \} }
{ \use:c \{ @@-normal-syntax \} }
\bool_if:NTF \l_@@_light_syntax_bool
{ \use:c \{ end @@-light-syntax \} }
{ \use:c \{ end @@-normal-syntax \} }
\c_math_toggle_token
\skip_horizontal:N \l_@@_left_margin_dim
\skip_horizontal:N \l_@@_extra_left_margin_dim
\c_math_toggle_token
\bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:
The preamble will be constructed in \g_@@_preamble_tl.
\@@_construct_preamble:n { #4 }
Now, the preamble is constructed in \g_@@_preamble_tl

Here is the beginning of the box which will contain the array. The \hbox_set_end: corresponding to this \hbox_set:Nw will be in the second part of the environment (and the closing \c_math_toggle_token also).
\hbox_set:Nw \l_the_array_box
\skip_horizontal:N \l_@@_left_margin_dim
\skip_horizontal:N \l_@@_extra_left_margin_dim
\c_math_toggle_token
\bool_if:NTF \l_@@_light_syntax_bool
{ \use:c \{ @@-light-syntax \} }
{ \use:c \{ @@-normal-syntax \} }
\c_math_toggle_token
\skip_horizontal:N \l_@@_right_margin_dim
\begin{align*}
\text{The array will be composed in a box (named } \l_@@_the_array_box \text{) because we have to do manipulations concerning the potential exterior rows.}

\box_clear_new:N \l_@@_the_array_box
\end{align*}

If the user has loaded nicematrix with the option define-L-C-R, he will be able to use L, C and R instead of l, c and r in the preambles of the environments of nicematrix (it’s a compatibility mode since L, C and R were mandatory before version 5.0).

\bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:
The preamble will be constructed in \g_@@_preamble_tl.
\@@_construct_preamble:n { #4 }

Now, the preamble is constructed in \g_@@_preamble_tl
End of the construction of the array (in the box \texttt{\l_@@_the_array_box}).

It the user has used the key \texttt{last-row} with a value, we control that the given value is correct (since we have just constructed the array, we know the real number of rows of the array).

\begin{verbatim}
\int_compare:nNnT \l_@@_last_row_int > {-2} 
{ \bool_if:NF \l_@@_last_row_without_value_bool 
  { \int_compare:nNnF \l_@@_last_row_int = \c@iRow 
    { \@@_error:n { Wrong-last-row } 
      \int_gset_eq:NN \l_@@_last_row_int \c@iRow 
    } 
  } 
}
\end{verbatim}

Now, the definition of \texttt{\c@jCol} and \texttt{\g_@@_col_total_int} change: \texttt{\c@jCol} will be the number of columns without the “last column”; \texttt{\g_@@_col_total_int} will be the number of columns with this “last column”.\footnote{We remind that the potential “first column” (exterior) has the number 0.}

\begin{verbatim}
\int_gset_eq:NN \c@jCol \g_@@_col_total_int 
\bool_if:nTF \g_@@_last_col_found_bool 
{ \int_gdecr:N \c@jCol } 
{ \int_compare:nNnT \l_@@_last_col_int > {-1} 
  { \@@_error:n { last-col-not-used } } 
} 
\bool_if:NF \l_@@_Matrix_bool 
{ \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int 
  { \@@_error:n { columns-not-used } } 
} 
\end{verbatim}

We fix also the value of \texttt{\c@iRow} and \texttt{\g_@@_row_total_int} with the same principle.

\begin{verbatim}
\int_compare:nNnT \l_@@_last_row_int > {-1} \{ \int_gdecr:N \c@iRow \} 
\int_gset_eq:NN \g_@@_row_total_int \c@iRow 
\end{verbatim}

Now, we begin the real construction in the output flow of \TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in \texttt{\g_@@_width_first_col_dim}; see p. 92).

\begin{verbatim}
\int_compare:nNnT \l_@@_first_col_int = 0 
{ \skip_horizontal:N \col@sep 
  \skip_horizontal:N \g_@@_width_first_col_dim 
} 
\end{verbatim}

The construction of the real box is different when \texttt{\l_@@_NiceArray_bool} is true (\{NiceArray\} or \{NiceTabular\}) and in the other environments because, in \{NiceArray\} or \{NiceTabular\}, we have no delimiter to put (but we have tabular notes to put). We begin with this case.

Remark that, in all cases, \texttt{\_use_arraybox_with_notes_c} is used.

\begin{verbatim}
\bool_if:nTF \l_@@_NiceArray_bool 
{ \str_case:VnF \l_@@_baseline_str 
  { b \l_@@_use_arraybox_with_notes_b: 
    c \l_@@_use_arraybox_with_notes_c: 
  } 
  \\_use_arraybox_with_notes: 
} 
\end{verbatim}

Now, in the case of an environment \texttt{\pNiceArray}, \texttt{\bNiceArray}, etc. We compute \texttt{\l_tmpa_dim} which is the total height of the “first row” above the array (when the key \texttt{first-row} is used).
\int_compare:nNnTF \l_@@_first_row_int = 0
\{
\dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
\dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
\}
\dim_zero:N \l_tmpa_dim

We compute \l_tmpb_dim which is the total height of the “last row” below the array (when the key \texttt{last-row} is used). A value of $-2$ for \l_@@_last_row_int means that there is no “last row.”\footnote{A value of $-1$ for \l_@@_last_row_int means that there is a “last row” but the user have not set the value with the option \texttt{last row} (and we are in the first compilation).}
\int_compare:nNnTF \l_@@_last_row_int > {-2}
\{
\dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
\dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
\}
\dim_zero:N \l_tmpb_dim
\hbox_set:Nn \l_tmpa_box
\{
\c_math_toggle_token
\left #1
\center
\}

We take into account the “first row” (we have previously computed its total height in \l_tmpa_dim). The \texttt{\hbox:n} (or \texttt{\hbox}) is necessary here.
\skip_vertical:N -\l_tmpa_dim
\skip_vertical:N -\arrayrulewidth
\hbox
\{
\bool_if:NTF \l_@@_NiceTabular_bool
\{
\skip_horizontal:N -\tabcolsep
\}
\skip_horizontal:N -\arraycolsep
\@@_use_arraybox_with_notes_c:
\bool_if:NTF \l_@@_NiceTabular_bool
\{
\skip_horizontal:N -\tabcolsep
\}
\skip_horizontal:N -\arraycolsep
\}

We take into account the “last row” (we have previously computed its total height in \l_tmpb_dim).
\skip_vertical:N -\l_tmpb_dim
\skip_vertical:N \arrayrulewidth
\right #2
\c_math_toggle_token
\}

Now, the box \l_tmpa_box is created with the correct delimiters.
We will put the box in the \TeX flow. However, we have a small work to do when the option \texttt{max-delimiter-width} is used.
\bool_if:NTF \l_@@_max_delimiter_width_bool
\{
\@@_put_box_in_flow_bis:nn { #1 } { #2 }
\}
\@@_put_box_in_flow:

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in \g_@@_width_last_col_dim: see p. \pageref{p}).
\bool_if:NTF \g_@@_last_col_found_bool
\{
\skip_horizontal:N \g_@@_width_last_col_dim
\skip_horizontal:N \col@sep
\}
\@@_after_array:
The aim of the following \texttt{\egroup} (the corresponding \texttt{\bgroup} is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

\begin{verbatim}
\egroup \\
bool_if:NT \c_@@_footnote_bool \endsavenotes
\end{verbatim}

This is the end of the environment \{NiceArrayWithDelims\}.

\section*{We construct the preamble of the array}

The transformation of the preamble is an operation in several steps.

The argument of \texttt{\@@\_construct\_preamble:n} is the preamble as given by the final user to the environment \{NiceTabular\} (or a variant). The preamble will be constructed in \texttt{\g_@@\_preamble_tl}.

\begin{verbatim}
\cs_new_protected:Npn \@@_construct_preamble:n #1
{ First, we will do an "expansion" of the preamble with the tools of the package array itself. This "expansion" will expand all the constructions with * and with all column types (defined by the user or by various packages using \texttt{\newcolumntype}). 
Since we use the tools of array to do this expansion, we will have a programmation which is not in the style of expl3.
We redefine the column types \texttt{w} and \texttt{W}. We use \texttt{\@@\_newcolumntype} instead of \texttt{\newcolumntype} because we don't want warnings for column types already defined. These redefinitions are in fact \texttt{protections} of the letters \texttt{w} and \texttt{W}. We don't want these columns type expanded because we will do the patch ourselves after. We want to be able the standard column types \texttt{w} and \texttt{W} in potential \{tabular\} of array in some cells of our array. That's why we do those redefinitions in a TeX group.
\begin{verbatim}
\if\@tempswa \fi { \the \NC@list }
\end{verbatim}
Now, we have to "patch" that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the "expansion") following by a marker \texttt{\q_stop} and we will consume these tokens constructing the (new form of the) preamble in \texttt{\g_@@\_preamble_tl}.
This is done recursively with the command \texttt{\@@\_patch\_preamble:n}. In the same time, we will count the columns with the counter \texttt{\c@\jCol}.
\begin{verbatim}
\int_gzero_new:N \c@\jCol
\bool_if:NTF \l_@@_Matrix_bool \\
{ \tl_gset:Nn \g_@@\_preamble_tl { \#1 } }
\end{verbatim}
\end{verbatim}

If we are in an environment without explicit preamble, we have nothing to do (excepted the treatment on both sides of the preamble which will be done at the end).

\begin{verbatim}
\bool_if:NTF \l_@@_Matrix_bool \\
{ \tl_gset:Nn \g_@@\_preamble_tl { \#1 } }
\end{verbatim}

\begin{verbatim}
\@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
\@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
\end{verbatim}

First, we have to store our preamble in the token register \texttt{\@temptokena} (those "token registers" are not supported by expl3).

\begin{verbatim}
\@temptokena { \#1 }
\end{verbatim}

Initialisation of a flag used by array to detect the end of the expansion.

\begin{verbatim}
\@tempswatrue
The following line actually does the expansion (it's has been copied from array.sty).
\begin{verbatim}
\@whilesw \if\tempswa \fi { \tempswafalse \the \NC@list }
\end{verbatim}
\end{verbatim}

Now, we have to "patch" that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker \texttt{\q_stop} and we will consume these tokens constructing the (new form of the) preamble in \texttt{\g_@@\_preamble_tl}.

\begin{verbatim}
\int_gzero_new:N \c@\jCol
\bool_if:NTF \l_@@_vlines_bool \\
{ \tl_gset:Nn \g_@@\_preamble_tl \\
{ \skip_horizontal:N \arrayrulewidth }
\end{verbatim}

79
The counter \l_tmpa_int will be count the number of consecutive occurrences of the symbole 1.
\int_zero:N \l_tmpa_int

Now, we actually patch the preamble (and it is constructed in \g_@@_preamble_tl).
\exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
\int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol \}

Now, we replace \columncolor by \@@_columncolor_preamble.
\bool_if:NT \l_@@_colortbl_like_bool
\{
\regex_replace_all:NnN \c_@@_columncolor_regex { \c { \@@_columncolor_preamble } } \g_@@_preamble_tl \}

We complete the preamble with the potential “exterior columns”.
\int_compare:nNnTF \l_@@_first_col_int = 0
\{ \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl \}
\{
\bool_lazy_all:nT \{
\l_@@_NiceArray_bool \{
\bool_not_p:n \l_@@_NiceTabular_bool \}
\bool_not_p:n \l_@@_vlines_bool \}
\bool_not_p:n \l_@@_exterior_arraycolsep_bool \}
\{ \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } \}
\}
\int_compare:nNnTF \l_@@_last_col_int > { -1 }
\{ \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl \}
\{
\bool_lazy_all:nT \{
\l_@@_NiceArray_bool \{
\bool_not_p:n \l_@@_NiceTabular_bool \}
\bool_not_p:n \l_@@_vlines_bool \}
\bool_not_p:n \l_@@_exterior_arraycolsep_bool \}
\{ \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } \}
\}

We add a last column to raise a good error message when the user put more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in \{NiceTabular*\} (\l_@@_tabular_width_dim=0pt).
\dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
\{ \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_error_too_much_cols: } l } \}

Now, we have to close the TeX group which was opened for the redefinition of the columns of type \texttt{w} and \texttt{W}.
\group_end:

\cs_new_protected:Npn \@@_patch_preamble:n #1
\{ \str_case:nnF { #1 } \{ #1 \}
\}

80
For `c`, `l` and `r`

\cs_new_protected:Npn \@@_patch_preamble_i:n #1
\{\tl_gput_right:Nn \g_@@_preamble_tl \{ > \{ \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } \} \} < \@@_end_Cell: \}\}

We increment the counter of columns.

\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n
\}

For `>`, `!` and `@`

\cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
\{\tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } } \@@_patch_preamble:n \}

For `|`

\cs_new_protected:Npn \@@_patch_preamble_iii:n #1
\{\tl_gput_right:Nn \g_@@_preamble_tl \{ > \{ \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } \} \} ! \} \}

\exp_not:N ! \skip_horizontal:n \}

\l_tmpa_int is the number of successive occurrences of `|`

\int_incr:N \l_tmpa_int
\@@_patch_preamble_iii_i:n
\}

\cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
\{\str_if_eq:nnTF \{ #1 \} \{ \@@_patch_preamble_iii:n | \}
\{ \tl_gput_right:Nx \g_@@_preamble_tl \exp_not:N ! \skip_horizontal:n \}

\l_tmpa_int
\dim_eval:n
{
  \arrayrulewidth * \l_tmpa_int
  + \doublerulesep * (\l_tmpa_int - 1)
}
}
\tl_gput_right:Nx \g_@@_internal_code_after_tl
{ \\vline:nn { \\succ:n \c@jCol } { \int_use:N \l_tmpa_int }
\int_zero:N \l_tmpa_int
\@@_patch_preamble:n #1
}

For p, m and b
\cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
{ \tl_gput_right:Nn \g_@@_preamble_tl
  { \hbox_set:Nw \l_@@_cell_box
    \@@_Cell:
    \tl_set:Nn \l_@@_cell_type_tl { #1 }
    \box_use:N \@arstrutbox
  }
  c
  < { \box_use:N \@arstrutbox \end { minipage } \@@_end_Cell: }
}
We increment the counter of columns.
\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n

For w and W
\cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
{ \tl_gput_right:Nn \g_@@_preamble_tl
  { \hbox_set:Nw \l_@@_cell_box
    \@@_Cell:
    \tl_set:Nn \l_@@_cell_type_tl { #1 }
  }
  c
  < { \@@_end_Cell:
    #1
    \bbox_set_end:
    \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
    \@@_adjust_width_box:
    \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
  }
}
We increment the counter of columns.
\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n

For \@@_true_c: which will appear in our redefinition of the columns of type S (of siunitx).
\cs_new_protected:Npn \@@_patch_preamble_vi:n #1
{ \tl_gput_right:Nn \g_@@_preamble_tl { c }
We increment the counter of columns.
\begin{verbatim}
\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n
\end{verbatim}

The command \@@_vdottedline:n is protected, and, therefore, won't be expanded before writing on \g_@@_internal_code_after_tl.
\begin{verbatim}
\tl_gput_right:Nn \g_@@_preamble_tl
\{ ! { \skip_horizontal:N 2\l_@@_radius_dim } \}
\tl_gput_right:Nx \g_@@_internal_code_after_tl
{ \@@_vdottedline:n \{ \int_use:N \c@jCol \} }
\@@_patch_preamble:n
\end{verbatim}

After a specifier of column, we have to test whether there is one or several <{...} because, after those potential <{...}, we have to insert !{\skip_horizontal:N ...} when the key vlines is used.
\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_viii:n #1
\{ \str_if_eq:nnTF { #1 } { < } \@@_patch_preamble_ix:n \}
\tl_gput_right:Nn \g_@@_preamble_tl
\{ ! { \skip_horizontal:N \arrayrulewidth } \}
\@@_patch_preamble:n { #1 }
\end{verbatim}

The command \@@_put_box_in_flow: puts the box \l_tmpa_box (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the depth to take back into account the potential exterior rows (the total height of the first row has been computed in \l_tmpa_dim and the total height of the potential last row in \l_tmpb_dim).
\begin{verbatim}
\cs_new_protected:Npn \@@_put_box_in_flow:
{ \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
\box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
\str_if_eq:VnTF \l_@@_baseline_str { c }
{ \box_use_drop:N \l_tmpa_box }
\@@_put_box_in_flow_i:
}
\end{verbatim}

The command \@@_put_box_in_flow_i: is used when the value of \l_@@_baseline_str is different of c (which is the initial value and the most used).
\begin{verbatim}
\cs_new_protected:Npn \@@_put_box_in_flow_i:
{ \pgfpicture
\@@_qpoint:n { row - 1 }
\dim_gset_eq:NN \g_tmpa_dim \pgfy
\@@_qpoint:n { row = \@@_succ:n \c@iRow }
\dim_gadd:Nn \g_tmpa_dim \pgfy
\dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }
\end{verbatim}

83
Now, $\g_{tmpa\_dim}$ contains the $y$-value of the center of the array (the delimiters are centered in relation with this value).

\begin{verbatim}
\str_if_in:NnTF \l_@@_baseline_str { line- }
  \{ \int_set:Nn \l_tmpa_int
  \{ \str_range:Nnn \l_@@_baseline_str 6
    \{ \str_count:N \l_@@_baseline_str \}
  \} \@@_qpoint:n \{ row - \int_use:N \l_tmpa_int \}
\}
\{ \str_case:VnF \l_@@_baseline_str
  \{ t \{ \int_set:Nn \l_tmpa_int 1 \}
  \{ b \{ \int_set_eq:NN \l_tmpa_int \c@iRow \}
  \}
  \{ \int_set:Nn \l_tmpa_int \l_@@_baseline_str \}
\bool_lazy_or:nnT
  \{ \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int \}
  \{ \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int \}
  \{ \@@_error:n \{ bad-value-for-baseline \}
  \int_set:Nn \l_tmpa_int 1
\}
\@@_qpoint:n \{ row - \int_use:N \l_tmpa_int - base \}
\end{verbatim}

We take into account the position of the mathematical axis.

\begin{verbatim}
\dim_gsub:Nn \g_{tmpa\_dim} { \fontdimen22 \textfont2 }
\end{verbatim}

Now, $\g_{tmpa\_dim}$ contains the value of the $y$ translation we have to do.

\begin{verbatim}
\endpgfpicture
\box_move_up:nn \g_{tmpa\_dim} \box_use_drop:N \l_tmpa_box
\box_use_drop:N \l_{tmpa\_box}
\end{verbatim}

The following command is always used by \texttt{NiceArrayWithDelims} (even if, in fact, there is no tabular notes: in fact, it's not possible to know whether there is or not before the composition of the blocks).

\begin{verbatim}
\cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
  \{
  \begin{minipage}[t]{\box_wd:N \l_@@_the_array_box}
  \box_use_drop:N \l_@@_the_array_box
  \end{minipage}
  \@@_create_extra_nodes:
  \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
  \bool_lazy_or:nnT
    \{ \int_compare_p:nNn \g_@@_blocks_seq > 0 \}
  \{ \tl_if_empty_p:V \l_@@_tabularnote_tl \}
  \@@_insert_tabularnotes:
  \end{verbatim}

We need a \texttt{minipage} because we will insert a LaTeX list for the tabular notes (that means that a \texttt{\vtop{\hsize=...}} is not enough).

\begin{verbatim}
\begin{minipage}[t]{\box_wd:N \l_@@_the_array_box}
\box_use_drop:N \l_@@_the_array_box
\end{verbatim}

We have to draw the blocks right now because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are medium nodes to create for the blocks.

\begin{verbatim}
\@@_create_extra_nodes:
\seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
\bool_lazy_or:nnT
  \{ \int_compare_p:nNn \g_@@_blocks_seq > 0 \}
  \{ \tl_if_empty_p:V \l_@@_tabularnote_tl \}
\@@_insert_tabularnotes:
\end{verbatim}

84
The \TeX{} group is for potential specifications in the \texttt{\_\_\_\_notes_code_before_tl}.

We compose the tabular notes with a list of \texttt{enumitem}. The \texttt{\textbackslash strut} and the \texttt{\textbackslash unskip} are designed to give the ability to put a \texttt{\textbackslash bottomrule} at the end of the notes with a good vertical space.

The following \texttt{\textbackslash par} is mandatory for the event that the user has put \texttt{\textbackslash footnotesize} (for example) in the notes/code-before.

The two dimensions \texttt{\textbackslash aboverulesep} and \texttt{\textbackslash heavyrulewidth} are parameters defined by \texttt{booktabs}.

\texttt{\_\_\_\_notes_code_after_tl} is the specification of color defined by \texttt{colortbl} but you use it even if \texttt{colortbl} is not loaded.

The case of \texttt{baseline} equal to \texttt{b}. Remember that, when the key \texttt{b} is used, the \texttt{\{}array\}\texttt{} (of \texttt{array}) is constructed with the option \texttt{t} (and not \texttt{b}). Now, we do the translation to take into account the option \texttt{b}.

\texttt{\_\_\_\_use_arraybox_with_notes_b:}
Now, the general case.

We convert a value of \( t \) to a value of \( 1 \).

\[
\text{str_if_eq:VnT \ l_@@_baseline_str \ t} \\
\{ \tl_set:Nn \ l_@@_baseline_str \ 1 \}
\]

Now, we convert the value of \( \l_@@_baseline_str \) (which should represent an integer) to an integer stored in \( \l_\text{tmpa_int} \).

\[
\text{pgfpicture} \\
\text{@_qpoint:n \{ \text{row} - 1 \}} \\
\text{dim_gset_eq:NN \ g_\text{tmpa_dim} \ \text{pgf@y}} \\
\text{str_if_in:NnTF \ l_@@_baseline_str \ line-} \\
\{ \\
\text{int_set:Nn \ l_\text{tmpa_int}} \\
\{ \\
\text{str_range:Nnn \ l_@@_baseline_str \ 6} \\
\{ \text{str_count:N \ l_@@_baseline_str} \\
\} \\
\text{@_qpoint:n \{ \text{row} - \text{int_use:N \ l_\text{tmpa_int}} \}} \\
\} \\
\{ \\
\text{int_set:Nn \ l_\text{tmpa_int} \ l_@@_baseline_str} \\
\text{bool_lazy_or:nnT} \\
\{ \text{int_compare_p:nNn \ l_\text{tmpa_int} < \ l_@@_first_row_int} \\
\{ \text{int_compare_p:nNn \ l_\text{tmpa_int} > \ g_\text{@@_row_total_int}} \\
\{ \\
\text{@_error:n \{ \text{bad-value-for-baseline} \}} \\
\text{int_set:Nn \ l_\text{tmpa_int} \ 1} \\
\} \\
\text{@_qpoint:n \{ \text{row} - \text{int_use:N \ l_\text{tmpa_int} - base} \}} \\
\} \\
\text{dim_gsub:NN \ g_\text{tmpa_dim} \ \text{pgf@y}} \\
\endpgfpicture
\]

The command \( \text{@_put_box_in_flow_bis:} \) is used when the option \text{max-delimiter-width} is used because, in this case, we have to adjust the widths of the delimiters. The arguments \#1 and \#2 are the delimiters specified by the user.

\[
\text{cs_new_protected:Npn \ @@_put_box_in_flow_bis:nn \ #1 \ #2} \\
\{ \\
\text{dim_zero_new:N \ l_@@_real_left_delim_dim} \\
\text{dim_zero_new:N \ l_@@_real_right_delim_dim} \\
\text{\hbox_set:Nn \ l_tmpb_box} \\
\{ \\
\text{c_math_toggle_token} \\
\text{left \ #1} \\
\text{vcenter} \\
\}
\]
Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.

\[ \left . \right . \]

The construction of the array in the environment \texttt{NiceArrayWithDelims} is, in fact, done by the environment \texttt{@@-light-syntax} or by the environment \texttt{@@-normal-syntax} (whether the option \texttt{light-syntax} is in force or not). When the key \texttt{light-syntax} is not used, the construction is a standard environment (and, thus, it’s possible to use verbatim in the array).

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it’s only a security). In order to detect whether it is empty, we test whether the next token is \texttt{\end} and, if it’s the case, we test if this is the end of the environment (if it is not, an standard error will be raised by \LaTeX{} for incorrect nested environments).

Here is the call to \texttt{array} (we have a dedicated macro \texttt{@@_array}: because of compatibility with the classes \texttt{revtex4-1} and \texttt{revtex4-2}).

When the key \texttt{light-syntax} is in force, we use an environment which takes its whole body as an argument (with the specifier \texttt{b} of \texttt{xparse}).

First, we test whether the environment is empty. It’s only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in \texttt{#1}.
Now, you extract the code-after of the body of the environment. Maybe, there is no command \CodeAfter in the body. That’s why you put a marker \CodeAfter after #1. If there is yet a \CodeAfter in #1, this second (or third...) \CodeAfter will be caught in the value of \g_nicematrix_code_after_tl. That doesn’t matter because \CodeAfter will be set to no-op before the execution of \g_nicematrix_code_after_tl.

Now, the second part of the environment. It is empty. That’s not surprising because we have caught the whole body of the environment with the specifier b provided by xparse.

The body of the array, which is stored in the argument #1, is now splitted into items (and not tokens).

If the environment uses the option last-row without value (i.e. without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list \l_@_code_for_last_row_tl is not empty, we will use directly where it should be.

Here is the call to \array (we have a dedicated macro \@_array: because of compatibility with the classes revtex4-1 and revtex4-2).

We need a global affectation because, when executing \l_tmpa_tl, we will exit the first cell of the array.

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it’s only a security).
We reput in the stream the \end{...} we have extracted and the user will have an error for incorrect nested environments.

The command \@@_create_col_nodes: will construct a special last row. That last row is a false row used to create the col nodes and to fix the width of the columns (when the array is constructed with an option which specify the width of the columns).

The following instruction must be put after the instruction \omit.

First, we put a col node on the left of the first column (of course, we have to do that after the \omit).

89
We compute in \texttt{\g_tmpa_skip} the common width of the columns (it’s a skip and not a dimension). We use a global variable because we are in a cell of an \texttt{\halign} and because we have to use this variable in other cells (of the same row). The affectation of \texttt{\g_tmpa_skip}, like all the affectations, must be done after the \texttt{\omit} of the cell.

We give a default value for \texttt{\g_tmpa_skip} (0 pt plus 1 fill) but it will just after be erased by a fixed value in the concerned cases.

\begin{verbatim}
\skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
\bool_if:NF \l_@@_auto_columns_width_bool
{ \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
{ \bool_lazy_and:nnTF \l_@@_auto_columns_width_bool
\l_@@_block_auto_columns_width_bool
{ \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim
\skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim
\skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
} \skip_horizontal:N \g_tmpa_skip
\hbox
\{ \bool_if:NT \l_@@_code_before_bool
{ \hbox
\{ \skip_horizontal:N -0.5\arrayrulewidth
\pgfsys@markposition \l_@@_env: - col - 2 }
\skip_horizontal:N \l_@@_columns_width_dim
\endpgfpicture
\endverbatim}

We begin a loop over the columns. The integer \texttt{\g_tmpa_int} will be the number of the current column. This integer is used for the Tikz nodes.

\begin{verbatim}
\int_gset:Nn \g_tmpa_int 1
\bool_if:NTF \g_@@_last_col_found_bool
{ \prg_replicate:nn \g_@@_col_total_int - 2 }
{ \prg_replicate:nn \g_@@_col_total_int - 1 }
\int_gincr:N \g_tmpa_int
\omit
\int_gincr:N \g_tmpa_int
\endverbatim

The incrementation of the counter \texttt{\g_tmpa_int} must be done after the \texttt{\omit} of the cell.
Here is the preamble for the “first column” (if the user uses the key `first-col`)

\tl_const:Nn \c_@@_preamble_first_col_tl
\{ 
  > 
  \@@_begin_of_row: 
  The contents of the cell is constructed in the box \l_@@_cell_box because we have to compute some dimensions of this box.
  \hbox_set:Nw \l_@@_cell_box
  \@@_math_toggle_token:
  \bool_if:NT \l_@@_small_bool \scriptstyle
  We insert \l_@@_code_for_first_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”.
\bool_lazy_and:nnT
{ \int_compare_p:nNn \c@iRow > 0 }
\bool_lazy_or_p:nn
{ \int_compare_p:nNn \l_@@_last_row_int < 0 }
{ \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
\l_@@_code_for_first_col_tl
\xglobal \colorlet { nicematrix-first-col } { . }

Be careful: despite this letter 1 the cells of the “first column” are composed in a \hbox_overlap_left:n.
l<
{ \@@_math_toggle_token:
  \hbox_set_end:
  \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
  \@@_adjust_width_box:
  \@@_update_for_first_and_last_row:
  \dim_gset:Nn \g_@@_width_first_col_dim
  { \dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box } }
  \hbox_overlap_left:n
  { \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
    \@@_node_for_the_cell:
    { \box_use_drop:N \l_@@_cell_box }
    \skip_horizontal:N \l_@@_left_delim_dim
    \skip_horizontal:N \l_@@_left_margin_dim
    \skip_horizontal:N \l_@@_extra_left_margin_dim
  }
  \bool_gset_false:N \g_@@_empty_cell_bool
  \skip_horizontal:N -2\col@sep
}

Here is the preamble for the “last column” (if the user uses the key last-col).
\tl_const:Nn \c_@@_preamble_last_col_tl
{ >
{ }

With the flag \g_@@_last_col_found_bool, we will know that the “last column” is really used.
\bool_gset_true:N \g_@@_last_col_found_bool
\int_gincr:N \c@jCol
\int_gset_eq:NN \g_@@_col_total_int \c@jCol

The contents of the cell is constructed in the box \l_tmpa_box because we have to compute some dimensions of this box.
\hbox_set:Nw \l_@@_cell_box
\@@_math_toggle_token:
\bool_if:NT \l_@@_small_bool \scriptstyle

We insert \l_@@_code_for_last_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”:
{ \int_compare_p:nNn \c@iRow > 0
  { \bool_lazy_or:nnT

92
We actualise the width of the “last column” because we will use this width after the construction of the array.

\dim_gset:Nn \g_@@_width_last_col_dim
{ \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
\skip_horizontal:N -2\col@sep

The content of the cell is inserted in an overlapping position.
\hbox_overlap_right:n
{ \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
{ \skip_horizontal:N \l_@@_right_delim_dim
\skip_horizontal:N \l_@@_right_margin_dim
\skip_horizontal:N \l_@@_extra_right_margin_dim
\@@_node_for_the_cell:
}
\bool_gset_false:N \g_@@_empty_cell_bool
}

The environment \{NiceArray\} is constructed upon the environment \{NiceArrayWithDelims\} but, in fact, there is a flag \l_@@_NiceArray_bool. In \{NiceArrayWithDelims\}, some special code will be executed if this flag is raised.
\NewDocumentEnvironment { NiceArray } { } { \bool_set_true:N \l_@@_NiceArray_bool
\str_if_empty:NT \g_@@_name_env_str { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
\@@_test_if_math_mode:
NiceArrayWithDelims . .
\endNiceArrayWithDelims }

We create the variants of the environment \{NiceArrayWithDelims\}.
\cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
{ \NewDocumentEnvironment { #1 NiceArray } { } { \str_if_empty:NT \g_@@_name_env_str { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
\@@_test_if_math_mode:
NiceArrayWithDelims #2 #3
}
The environment \{NiceMatrix\} and its variants

\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
{\bool_set_true:N \l_@@_Matrix_bool \use:c { #1 NiceArray } \begin{NiceArray}{*}{#2}}
\clist_map_inline:nn { { } , p , b , B , v , V }
{\NewDocumentEnvironment { #1 NiceMatrix } { ! O { } } \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix } \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 } \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl \use:c { end #1 NiceArray } }\endNiceArrayWithDelims}

The environments \{NiceTabular\} and \{NiceTabular*\}

\NewDocumentEnvironment { NiceTabular } { O { } m ! O { } } { \str_gset:Nn \g_@@_name_env_str { NiceTabular } \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 } \bool_set_true:N \l_@@_NiceTabular_bool \NiceArray { #2 } \use:c { end #1 NiceArray } }

\NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } } { \str_gset:Nn \g_@@_name_env_str { NiceTabular* } \dim_set:Nn \l_@@_tabular_width_dim { #1 } \keys_set:nn { NiceMatrix / NiceTabular* } { #2 , #4 } \bool_set_true:N \l_@@_NiceTabular_bool \NiceArray { #3 } \use:c { end #1 NiceArray } }
After the construction of the array

\cs_new_protected:Npn \@@_after_array:
\group_begin:

When the option \texttt{last-col} is used in the environments with explicit preambles (like \texttt{NiceArray}, \texttt{pNiceArray}, etc.) a special type of column is used at the end of the preamble in order to compose the cells in an overlapping position (with \texttt{\hbox_overlap_right:n}) but (if \texttt{last-col} has been used), we don’t have the number of that last column. However, we have to know that number for the color of the potential \texttt{\Vdots} drawn in that last column. That’s why we fix the correct value of \texttt{\l_@@_last_col_int} in that case.

\bool_if:NT \g_@@_last_col_found_bool
\{ \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int \}

If we are in an environment without preamble (like \texttt{NiceMatrix} or \texttt{pNiceMatrix}) and if the option \texttt{last-col} has been used without value we fix the real value of \texttt{\l_@@_last_col_int}.

\bool_if:NT \l_@@_last_col_without_value_bool
\{ \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int \}

If the option \texttt{light-syntax} is used, we have nothing to write since, in this case, the number of rows is directly determined.

\bool_if:NF \l_@@_light_syntax_bool
\{ \dim_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int \}

If the environment has a name, we also write a value based on the name because it’s more reliable than a value based on the number of the environment.
If the key `code-before` is used, we have to write on the `aux` file the actual size of the array.

```latex
\bool_if:NT \l_@@_code_before_bool
\{\protect\begin{lrbox}{\mainaux}\ExplSyntaxOn\protect\end{lrbox}\noexpand\int_use:N \g_@@_env_int
\{\seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } \}
\bool_if:NT \l_@@_code_before_bool
\noexpand\setbox0\usebox\mainaux\noexpand\setbox\mainaux\usebox0\ExplSyntaxOff\protect\end{lrbox}
```

If the user has used a key `last-row` in an environment with preamble (like `{pNiceArray}`) and that last row has not been found, we have to increment the value because it will be decreased when used in the `code-before`.

```latex
\bool_lazy_and:nnTF{\int_compare_p:nNn \l_@@_last_col_int > { -2 } }{\bool_not_p:n \g_@@_last_col_found_bool}
\@@_succ:n\int_use:N\g_@@_col_total_int
```

We write also the potential content of \g_@@_pos_of_blocks_seq (it will be useful if the commands \rowcolors is used with the key respect-blocks.

```latex
\seq_gset_from_clist:cn{ c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq }
\seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , 
```

By default, the diagonal lines will be parallelized. There are two types of diagonals lines: the \Ddots diagonals and the \Iddots diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

The dimensions \g_@@_delta_x_one_dim and \g_@@_delta_y_one_dim will contain the $\Delta_x$ and $\Delta_y$ of the first \Ddots diagonal. We have to store these values in order to draw the others \Ddots diagonals parallel to the first one. Similarly \g_@@_delta_x_two_dim and \g_@@_delta_y_two_dim are the $\Delta_x$ and $\Delta_y$ of the first \Iddots diagonal.

If the option `small` is used, the values \l_@@_radius_dim and \l_@@_inter_dots_dim (used to draw the dotted lines created by \hdottedline and \vdotteline and also for all the other dotted lines when line-style is equal to standard, which is the initial value) are changed.

```latex
\int_zero_new:N \l_@@_initial_i_int
\int_zero_new:N \l_@@_initial_j_int
\int_zero_new:N \l_@@_final_i_int
\int_zero_new:N \l_@@_final_j_int
\bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
```

39 It's possible to use the option parallelize-diags to disable this parallelization.
The dimension \l_@@_xdots_shorten_dim corresponds to the option xdots/shorten available to the user. That’s why we give a new value according to the current value, and not an absolute value.

Now, we actually draw the dotted lines (specified by \Cdots, \Vdots, etc.).

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it will do nothing.

The following code is only for efficiency. We determine whether the potential horizontal and vertical rules are “complete”, that is to say drawn in the whole array. We are sure that all the rules will be complete when there is no block, no virtual block (determined by a command such as \Cdots, \Vdots, etc.) and no corners. In that case, we switch to a shortcut version of \@@_vline_i:nn and \@@_hline:nn.

Now, the code-after.

When light-syntax is used, we insert systematically a \CodeAfter in the flow. Thus, it’s possible to have two instructions \CodeAfter and the second may be in \g_nicematrix_code_after_tl. That’s why we set \CodeAfter to be no-op now.

And here’s the code-after:

\g_nicematrix_code_before_tl is for instructions in the cells of the array such as \rowcolor and \cellcolor (when the key colortbl-like is in force). These instructions will be written on the aux file to be added to the code-before in the next run.
The command \rowcolor in tabular will in fact use \rectanglecolor in order to follow the behaviour of \rowcolor of colorbl. That’s why there may be a command \rectanglecolor in \g_nicematrix_code_before_tl. In order to avoid an error during the expansion, we define a protected version of \rectanglecolor.

\cs_set_protected:Npn \rectanglecolor { }
\cs_set_protected:Npn \columncolor { }
\iow_now:Nn \@mainaux \ExplSyntaxOn
\tl_gset:cn { g_@@_code_before_ \int_use:N \g_@@_env_int_tl }
{ \g_nicematrix_code_before_tl }
\iow_now:Nn \@mainaux \ExplSyntaxOff
\bool_set_true:N \l_@@_code_before_bool
\str_gclear:N \g_@@_name_env_str
\@@_restore_iRow_jCol:

The command \CT@arc@ contains the instruction of color for the rules of the array. This command is used by \CT@arc@ but we use it also for compatibility with colorbl. But we want also to be able to use color for the rules of the array when colorbl is not loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by colorbl.

\cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@

We recall that, when externalization is used, \tikzpicture and \endtikzpicture (or \pgfpicture and \endpgfpicture) must be directly “visible”. That’s why we have to define the adequate version of \@@_draw_dotted_lines: whether Tikz is loaded or not (in that case, only PGF is loaded).

\AtBeginDocument
\cs_new_protected:Npx \@@_draw_dotted_lines:
\{ \@@_draw_dotted_lines_i:
\} \@@_restore_iRow_jCol:

The following command must be protected because it will appear in the construction of the command \@@_draw_dotted_lines:.

\cs_new_protected:Npn \@@_draw_dotted_lines_i:
\{ \pgfrememberpicturepositiononpagetrue \pgf@relevantforpicturesizefalse \g_@@_HVdots_for_lines_tl \g_@@_Vdots_lines_tl \g_@@_Ddots_lines_tl \g_@@_Iddots_lines_tl \g_@@_Cdots_lines_tl \g_@@_Ldots_lines_tl \}

\cs_new_protected:Npn \@@_restore_iRow_jCol:
\{ \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int } \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int } \}

\textsuperscript{40} e.g. \color\{rgb\}{0.5,0.5,0)
We draw the dotted lines

A dotted line will be said open in one of its extremities when it stops on the edge of the matrix and closed otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

\[
\begin{pmatrix}
 a + b + c & a + b & a \\
 a \cdots \cdots \cdots \cdots \\
 a & a + b & a + b + c
\end{pmatrix}
\]

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the \(x\)-value of the orientation vector of the line;
- the fourth argument is the \(y\)-value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

\begin{verbatim}
\cs_new_protected:Npn \@@_find_extremities_of_line:nnnn \#1 \#2 \#3 \#4
{
\l_@@_initial_i_int \l_@@_initial_j_int \l_@@_final_i_int \l_@@_final_j_int
\l_@@_initial_open_bool \l_@@_final_open_bool
\int_set:Nn \l_@@_initial_i_int { \#1 }
\int_set:Nn \l_@@_initial_j_int { \#2 }
\int_set:Nn \l_@@_final_i_int { \#1 }
\int_set:Nn \l_@@_final_j_int { \#2 }
\bool_set_false:N \l_@@_stop_loop_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
{\int_add:Nn \l_@@_final_i_int { \#3 } \int_add:Nn \l_@@_final_j_int { \#4 }}
\bool_set_false:N \l_@@_final_open_bool
\int_compare:nNnTF \l_@@_final_i_int > \c@iRow
{\bool_set_true:N \l_@@_final_open_bool }
\int_compare:nNnTF \l_@@_final_j_int > \c@jCol
{\bool_set_true:N \l_@@_final_open_bool }
\bool_set_false:N \l_@@_final_open_bool
\int_compare:nNnTF \l_@@_final_j_int < 1
{\bool_set_true:N \l_@@_final_open_bool }
\end{verbatim}
If we are outside the matrix, we have found the extremity of the dotted line and it’s an open extremity.


We do a step backwards.

If we are in the matrix, we test whether the cell is empty. If it’s not the case, we stop the loop because we have found the correct values for \l_\@final_i_int and \l_\@final_j_int.


If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environnement), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.
For \l@@_initial_i_int and \l@@_initial_j_int the programmation is similar to the previous one.

```latex
\bool_set_false:N \l@@_stop_loop_bool
\bool_do_until:Nn \l@@_stop_loop_bool
\int_sub:Nn \l@@_initial_i_int \{ #3 \}
\int_sub:Nn \l@@_initial_j_int \{ #4 \}
\bool_set_false:N \l@@_initial_open_bool
\int_compare:nNnTF \l@@_initial_i_int \{ 1 \}
\int_compare:nNnTF \l@@_initial_j_int \{ 0 \}
\bool_set_true:N \l@@_initial_open_bool
\int_add:Nn \l@@_initial_i_int \{ #3 \}
\int_add:Nn \l@@_initial_j_int \{ #4 \}
\bool_set_true:N \l@@_stop_loop_bool
\cs_if_exist:cTF
\int_use:N \l@@_initial_i_int - \int_use:N \l@@_initial_j_int
\int_add:Nn \l@@_initial_i_int \{ #3 \}
\int_add:Nn \l@@_initial_j_int \{ #4 \}
\bool_set_true:N \l@@_initial_open_bool
\bool_set_true:N \l@@_stop_loop_bool
\cs_set:cpn
\int_use:N \l@@_initial_i_int - 101
```

101
We remind the rectangle described by all the dotted lines in order to respect the corresponding virtual “block” when drawing the horizontal and vertical rules.

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.
The command \@@_actually_draw_Ldots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

The following function is also used by \Hdotsfor.

\cs_new_protected:Npn \@@_actually_draw_Ldots:nnn { #1 #2 #3 }
{
 \bool_if:NTF \l_@@_initial_open_bool
 { \@@_qpoint:n { \int_use:N \l_@@_initial_j_int }
   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
   \dim_add:Nn \l_@@_x_initial_dim \col@sep
   \@@_qpoint:n { \row_use:N \l_@@_initial_i_int - base }
   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
 } { \@@_set_initial_coords_from_anchor:n { base~east } }
 \bool_if:NTF \l_@@_final_open_bool
 { \@@_qpoint:n { \@@_succ:n \int_use:N \l_@@_final_j_int }
   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
   \dim_sub:Nn \l_@@_x_final_dim \col@sep
   \@@_qpoint:n { \row_use:N \l_@@_final_i_int - base }
   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
 } { \@@_set_final_coords_from_anchor:n { base~west } }
 \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
 \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
 \@@_draw_line:nnn #1 #2 #3
}

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option line-style is used (?)..

\dim_add:NNn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_add:NNn \l_@@_y_final_dim \l_@@_radius_dim
\@@_draw_line:nnn #1 #2 #3

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\group_begin:
\int_compare:nNnTF { #1 } = 0
{ \color { nicematrix-first-row } }
{
\keys_set:nN { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:
}

We remind that, when there is a “last row” \l_@@_last_row_int will always be (after the construction of the array) the number of that “last row” even if the option last-row has been used without value.

\int_compare:nNnTF { #1 } = \l_@@_last_row_int
{ \color { nicematrix-last-row } }
{
\keys_set:nN { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:
}

The command \@@_actually_draw_Cdots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.
The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

The command \@@_actually_draw_Vdots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

The following function is also used by \Vdotsfor.

Now, we try to determine whether the column is of type c or may be considered as if.
We may think that the final user won’t use a “last column” which contains only a command \Vdots. However, if the \Vdots is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

Now the case where both extremities are closed. The first conditional tests whether the column is of type c (C of \{NiceArray\}) or may be considered as if.

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\group_begin:
  \keys_set:nn { NiceMatrix / xdots } { #3 }
  \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Ddots:
  \group_end:

The command \@@_actually_draw_Ddots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

\cs_new_protected:Npn \@@_actually_draw_Ddots:
  { \bool_if:NTF \l_@@_initial_open_bool
    { \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int } \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int } \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x \} \{ \@@_set_initial_coords_from_anchor:n \} \} \bool_if:NT \l_@@_final_open_bool
  { \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int } \dim_set_eq:NN \l_@@_y_final_dim \pgf@y \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int } \dim_set_eq:NN \l_@@_x_final_dim \pgf@x \} \{ \@@_set_final_coords_from_anchor:n \} \} \bool_if:NT \l_@@_parallelize_diags_bool
  { \int_gincr:N \g_@@_ddots_int \int_compare:nNnTF \g_@@_ddots_int = 1
    { \dim_gset:Nn \g_@@_delta_x_one_dim \l_@@_x_final_dim - \l_@@_x_initial_dim \dim_gset:Nn \g_@@_delta_y_one_dim \l_@@_y_final_dim - \l_@@_y_initial_dim \} } \int_gincr:N \g_@@_ddots_int

We have retrieved the coordinates in the usual way (they are stored in \l_@@_x_initial_dim, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

\bool_if:NT \l_@@_parallelize_diags_bool
  { \int_gincr:N \g_@@_ddots_int \int_compare:nNnTF \g_@@_ddots_int = 1
    { \dim_gset:Nn \g_@@_delta_x_one_dim \l_@@_x_final_dim - \l_@@_x_initial_dim \dim_gset:Nn \g_@@_delta_y_one_dim \l_@@_y_final_dim - \l_@@_y_initial_dim \} }
If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying
the coordinate \l_@@_x_initial_dim.

\begin{verbatim}
\dim_set:Nn \l_@@_y_final_dim
  { \l_@@_y_initial_dim +
      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) * \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim }
\end{verbatim}

We draw the \Iddots diagonals in the same way.
The first and the second arguments are the coordinates of the cell where the command has been
issued. The third argument is the list of the options.

\begin{verbatim}
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
  { \cs_if_free:cT { @@ _ dotted _ #1 - #2 } { \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 } } \keys_set:nn { NiceMatrix / xdots } { #3 } \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } } \@@_actually_draw_Iddots: } \group_end: }
\end{verbatim}

The command \@@_actually_draw_Iddots: has the following implicit arguments:

• \l_@@_initial_i_int
• \l_@@_initial_j_int
• \l_@@_initial_open_bool
• \l_@@_final_i_int
• \l_@@_final_j_int
• \l_@@_final_open_bool.

\begin{verbatim}
\cs_new_protected:Npn \@@_actually_draw_Iddots:
  { \bool_if:NTF \l_@@_initial_open_bool
      { \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int } \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
          \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int } \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      } \tl_if_empty:F \l_@@_set_initial_coords_from_anchor:n { south~west } { \bool_if:NTF \l_@@_final_open_bool
      { \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int } \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
          \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int } \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
      } }
\end{verbatim}
The actual instructions for drawing the dotted line with Tikz

The command \@@_draw_line: should be used in a \{pgfpicture\}. It has six implicit arguments:

- \l_@@_x_initial_dim
- \l_@@_y_initial_dim
- \l_@@_x_final_dim
- \l_@@_y_final_dim
- \l_@@_initial_open_bool
- \l_@@_final_open_bool

\cs_new_protected:Npn \@@_draw_line:
{\pgfrememberpicturepositiononpagetrue\pgf@relevantforpicturesizefalse\tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl \@@_draw_standard_dotted_line:
\@@_draw_non_standard_dotted_line:}

We have to do a special construction with \exp_args:NV to be able to put in the list of options in the correct place in the Tikz instruction.

\cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
{\begin {scope}\exp_args:No \@@_draw_non_standard_dotted_line:n {\l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl}\end {scope}}

We have used the fact that, in PGF, un color name can be put directly in a list of options (that’s why we have put directly \l_@@_xdots_color_tl).

The argument of \@@_draw_non_standard_dotted_line:n is, in fact, the list of options.

\cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
{\begin {scope}\exp_args:No \@@_draw_non_standard_dotted_line:n {\l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl} #1\end {scope}}
The command \@@_draw_standard_dotted_line: draws the line with our system of points (which give a dotted line with real round points).

\cs_new_protected:Npn \@@_draw_standard_dotted_line:
{
  First, we put the labels.

  \bool_lazy_and:nnF
  { \tl_if_empty_p:N \l_@@_xdots_up_tl }
  { \tl_if_empty_p:N \l_@@_xdots_down_tl }
  {
    \pgfscope
    \pgfonlayer{axis}
    \pgftransformshift
    \pgfpackagearray
    \pgfpackagearray{\l_@@_x_initial_dim}{\l_@@_y_initial_dim}
    \pgfpackagearray{\l_@@_x_final_dim}{\l_@@_y_final_dim}
    \pgfonlayer{axis}
    \pgfonlayer{main}
    \pgfnode{rectangle}{south}{\l_@@_xdots_up_tl}
    \pgfnode{rectangle}{north}{\l_@@_xdots_down_tl}
  }
}

\end{scope}
The dimension \( l_{\text{dim}} \) is the length \( \ell \) of the line to draw. We use the floating point reals of expl3 to compute this length.

\[ l_{\text{dim}} = \frac{\sqrt{2}}{2} (l_{\text{final}} - l_{\text{initial}})^2 + \left(\frac{y_{\text{final}} - y_{\text{initial}}}{2}\right) \]

It seems that, during the first compilations, the value of \( l_{\text{dim}} \) may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

The integer \( \text{int} \) is the number of dots of the dotted line.

\[ \text{int} = \frac{l_{\text{dim}}}{l_{\text{inter}}} \]


The dimensions $\l_\text{tmpa}_\dim$ and $\l_\text{tmpb}_\dim$ are the coordinates of the vector between two dots in the dotted line.

The length $\ell$ is the length of the dotted line. We note $\Delta$ the length between two dots and $n$ the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0$, 1 or 2. We first compute this number $k$ in $\l_\text{tmpb}_\int$.

In the loop over the dots, the dimensions $\l_\text{tmpa}_\dim$ and $\l_\text{tmpb}_\dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.
User commands available in the new environments

The commands \@@_Ldots, \@@_Cdots, \@@_Vdots, \@@_Ddots and \@@_Iddots will be linked to \Ldots, \Cdots, \Vdots, \Ddots and \Iddots in the environments \{NiceArray\} (the other environments of \nicematrix rely upon \{NiceArray\}).

The starred versions of these commands are deprecated since version 3.1 but, as of now, they are still available with an error.

The syntax of these commands uses the character _ as embellishment and that's why we have to insert a character _ in the \texttt{arg spec} of these commands. However, we don’t know the future catcode of _ in the main document (maybe the user will use \texttt{underscore}, and, in that case, the catcode is 13 because \texttt{underscore} activates _). That’s why these commands will be defined in a \AtBeginDocument and the \texttt{arg spec} will be rescanned.

\AtBeginDocument

\begin{Verbatim}
\tl_set:Nn \l_@@_argspec_tl \{ O { } E { _ ^ } { { } { } } \}
\tl_set_rescan:Nno \l_@@_argspec_tl \{ } \l_@@_argspec_tl
\exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
\{ \int_compare:nNnTF \c@jCol = 0
\{ \@@_error:nn \{ in-first-col \} \Ldots \}
\{ \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
\{ \@@_error:nn \{ in-last-col \} \Ldots \}
\{ \@@_instruction_of_type:nnn \c_false_bool \{ \ldots \}
\{ #1 , down = #2 , up = #3 \}
\}
\bool_if:NF \l_@@_nullify_dots_bool \{ \phantom \@@_old_ldots \}
\bool_gset_true:N \g_@@_empty_cell_bool
\}
\exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
\{ \int_compare:nNnTF \c@jCol = 0
\{ \@@_error:nn \{ in-first-col \} \Cdots \}
\{ \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
\{ \@@_error:nn \{ in-last-col \} \Cdots \}
\{ \@@_instruction_of_type:nnn \c_false_bool \{ \Cdots \}
\{ #1 , down = #2 , up = #3 \}
\}
\bool_if:NF \l_@@_nullify_dots_bool \{ \phantom \@@_old_cdots \}
\bool_gset_true:N \g_@@_empty_cell_bool
\}
\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
\{ \int_compare:nNnTF \c@iRow = 0
\{ \@@_error:nn \{ in-first-row \} \Vdots \}
\{ \@@_instruction_of_type:nnn \c_false_bool \{ \Vdots \}
\{ #1 , left = #2 , right = #3 \}
\}
\bool_if:NF \l_@@_nullify_dots_bool \{ \phantom \@@_old_vdots \}
\bool_gset_true:N \g_@@_empty_cell_bool
\}
\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
\{ \int_compare:nNnTF \c@iRow = 0
\{ \@@_error:nn \{ in-first-row \} \Ddots \}
\{ \@@_instruction_of_type:nnn \c_false_bool \{ \Ddots \}
\{ #1 , left = #2 , right = #3 \}
\}
\bool_if:NF \l_@@_nullify_dots_bool \{ \phantom \@@_old_ddots \}
\bool_gset_true:N \g_@@_empty_cell_bool
\}
\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
\{ \int_compare:nNnTF \c@iRow = 0
\{ \@@_error:nn \{ in-first-row \} \Iddots \}
\{ \@@_instruction_of_type:nnn \c_false_bool \{ \Iddots \}
\{ #1 , left = #2 , right = #3 \}
\}
\bool_if:NF \l_@@_nullify_dots_bool \{ \phantom \@@_old_iddots \}
\bool_gset_true:N \g_@@_empty_cell_bool
\}
\end{Verbatim}
\{ 
\int_compare:nNnTF \c@iRow = \l_@@_last_row_int 
\{ \@@_error:nn { in-last-row } \Vdots \} 
\{ 
\@@_instruction_of_type:nnn \c_false_bool { \Vdots } 
\{ #1 , down = #2 , up = #3 \} 
\} 
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_vdots } 
\bool_gset_true:N \g_@@_empty_cell_bool 
\}

\exp_args:NV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl 
\{ 
\int_case:nnF \c@iRow 
\{ 
0 { \@@_error:nn { in-first-row } \Ddots } 
\l_@@_last_row_int { \@@_error:nn { in-last-row } \Ddots } 
\} 
\{ 
\int_case:nnF \c@jCol 
\{ 
0 { \@@_error:nn { in-first-col } \Ddots } 
\l_@@_last_col_int { \@@_error:nn { in-last-col } \Ddots } 
\} 
\{ 
\keys_set_known:nn { NiceMatrix / Ddots } { #1 } 
\@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots } 
\{ #1 , down = #2 , up = #3 \} 
\} 
\} 
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ddots } 
\bool_gset_true:N \g_@@_empty_cell_bool 
\}

\exp_args:NV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl 
\{ 
\int_case:nnF \c@iRow 
\{ 
0 { \@@_error:nn { in-first-row } \Iddots } 
\l_@@_last_row_int { \@@_error:nn { in-last-row } \Iddots } 
\} 
\{ 
\int_case:nnF \c@jCol 
\{ 
0 { \@@_error:nn { in-first-col } \Iddots } 
\l_@@_last_col_int { \@@_error:nn { in-last-col } \Iddots } 
\} 
\{ 
\keys_set_known:nn { NiceMatrix / Ddots } { #1 } 
\@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots } 
\{ #1 , down = #2 , up = #3 \} 
\} 
\} 
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_iddots } 
\bool_gset_true:N \g_@@_empty_cell_bool 
\}

End of the \AtBeginDocument.
Despite its name, the following set of keys will be used for \Ddots but also for \iddots.

\keys_define:nn { NiceMatrix / Ddots }
  { draw-first .bool_set:N = \l_@@_draw_first_bool ,
    draw-first .default:n = true ,
    draw-first .value_forbidden:n = true }

The command \@@_Hspace: will be linked to \hspace in \{NiceArray\}.
\cs_new_protected:Npn \@@_Hspace:
  { \bool_gset_true:N \g_@@_empty_cell_bool \hspace }

In the environment \{NiceArray\}, the command \multicolumn will be linked to the following command \@@_multicolumn:nnn.
\cs_set_eq:NN \@@_old_multicolumn \multicolumn
\cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
  { % \begin{macrocode}
  % We have to act in an expandable way since it will begin by a |\multicolumn|.
  % \end{macrocode}
  \exp_args:NNe \@@_old_multicolumn { #1 }
  { \peek_remove_spaces:n { \tl_lower_case:n { #2 } } { #3 }
    { l | } { > \@@_Cell: l < \@@_end_Cell: | }
    { r | } { > \@@_Cell: r < \@@_end_Cell: | }
    { c | } { > \@@_Cell: c < \@@_end_Cell: | }
    { l | } { > \@@_Cell: l < \@@_end_Cell: | }
    { r | } { > \@@_Cell: r < \@@_end_Cell: | }
    { c | } { > \@@_Cell: c < \@@_end_Cell: | }
    { l | } { > \@@_Cell: l < \@@_end_Cell: | }
    { r | } { > \@@_Cell: r < \@@_end_Cell: | }
    { c | } { > \@@_Cell: c < \@@_end_Cell: | }
    { l | } { > \@@_Cell: l < \@@_end_Cell: | }
    { r | } { > \@@_Cell: r < \@@_end_Cell: | }
    { c | } { > \@@_Cell: c < \@@_end_Cell: | }
  }
\peek_remove_spaces:n

The \peek_remove_spaces:n is mandatory.
\peek_remove_spaces:n
  \{ \int_compare:nNnT #1 > 1
  { \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
    { \int_use:N \c@Row - \int_use:N \c@jCol } \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
  \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
  { \int_use:N \c@iRow + \c@jCol + #1 - 1 } \}
}
The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. Tikz nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must not be protected since it begins with `\multicolumn`.

The command `\@@_Hdotsfor_i` is defined with the tools of `xparse` because it has an optional argument. Note that such a command defined by `\NewDocumentCommand` is protected and that's why we have put the `\multicolumn` before (in the definition of `\@@_Hdotsfor:`).

```
\AtBeginDocument
\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
\prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
\AtBeginDocument.
```

For the row, it's easy.

```
\int_set:Nn \l_@@_initial_i_int { #1 }
\int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
\int_compare:nNnTF #2 = 1
\int_set:Nn \l_@@_initial_j_int 1
\bool_set_true:N \l_@@_initial_open_bool
```
We declare all the cells concerned by the \Hdotsfor as “dotted” (for the dotted lines created by \Cdots, \Ldots, etc., this job is done by \@@_find_extremities_of_line:n.nn). This declaration is done by defining a special control sequence (to nil).
\begin{Verbatim}[commandchars=\[\]]
\begin{verbatim}
  \def\mycommand{\exp_not:n { #4 }, up = \exp_not:n { #5 }}
\end{verbatim}
\end{Verbatim}

Enf of \verb|\AtBeginDocument|.

\begin{Verbatim}[commandchars=\[\]]
\begin{verbatim}
  \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
  {\begin{verbatim}
  \bool_set_false:N \l_@@_initial_open_bool
  \bool_set_false:N \l_@@_final_open_bool
  \foreach \column in {#2} {
    \int_set:Nn \l_@@_initial_j_int { \column }
    \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
  }
  \foreach \row in {1, #1} {
    \int_compare:nNnTF \row = 1
      { \int_set:Nn \l_@@_initial_i_int 1
        \bool_set_true:N \l_@@_initial_open_bool
      }
      { \cs_if_exist:cTF { \pgf@sh@ns@\@@_env: - \int_eval:n { \row - 1 } - \int_use:N \l_@@_initial_j_int } {
        \int_set:Nn \l_@@_initial_i_int { \row - 1 }
        \bool_set_true:N \l_@@_initial_open_bool
      } { \int_set:Nn \l_@@_initial_i_int { \row }
        \bool_set_true:N \l_@@_initial_open_bool
      } }
    \int_compare:nNnTF { \row + #3 - 1 } = \c@iRow
      { \color { \nicematrix-first-col } }
      { \cs_if_exist:cTF { \pgf@sh@ns@\@@_env: - \int_eval:n { \row + #3 } - \int_use:N \l_@@_final_j_int } {
        \int_set:Nn \l_@@_final_i_int { \row + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      } { \int_set:Nn \l_@@_final_i_int { \row + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      } }
  }
  \keys_set:nn { NiceMatrix / xdots } { #4 }
  \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
  \@@_actually_draw_Vdots:
\end{verbatim}
\end{verbatim}
\end{Verbatim
We declare all the cells concerned by the \Vdotsfor as “dotted” (for the dotted lines created by \Cdots, \Ldots, etc., this job is done by \@@_find_extremities_of_line:nnnn). This declaration is done by defining a special control sequence (to nil).

\int_step_inline:nnn { #1 } { #1 + #3 - 1 }
\{ \cs_set:cpn { @@_dotted _ ##1 - #2 } { } \}
}

The command \@@_rotate: will be linked to \rotate in {NiceArrayWithDelims}.
\cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

The command \line accessible in code-after

In the code-after, the command \@@_line:nn will be linked to \line. This command takes two arguments which are the specifications of two cells in the array (in the format i-j) and draws a dotted line between these cells.

First, we write a command with an argument of the format i-j and applies the command \int_eval:n to i and j; this must not be protected (and is, of course fully expandable).

\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
{ \int_eval:n { #1 } - \int_eval:n { #2 } }

With the following construction, the command \@@_double_int_eval:n is applied to both arguments before the application of \@@_line_i:nn (the construction uses the fact the \@@_line_i:nn is protected and that \@@_double_int_eval:n is fully expandable).

\AtBeginDocument
{\tl_set:Nn \l_@@_argspec_tl { O { } m m ! O { } E { _ ^ } { { } { } } }
\tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
\exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
{\group_begin:
\keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\use:e
{ \@@_line_i:nn
\int_step_inline:nnn { #2 } { #2 + #3 - 1 }
\{ \@@_double_int_eval:n #2 \q_stop
{ \@@_double_int_eval:n #3 \q_stop
}
\group_end:
}
\cs_new_protected:Npn \@@_line_i:nn #1 #2
{ \bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
\bool_if:nTF
{ \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
\|\|
\cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }

\textsuperscript{41}Indeed, we want that the user may use the command \line in code-after with LaTeX counters in the arguments — with the command \value.
We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:`.

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

Colors of cells, rows and columns

In the beginning of the code-before, the command `\@@_rowcolor:nn` will be linked to `\rowcolor` and the command `\@@_columncolor:nn` to `\columncolor`.

Here an example: `\@@_rowcolor {red!15} {1,3,5-7,10-}`

\l_tma_dim is the x-value of the right side of the rows.
Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`

\begin{itemize}
\item \l_tmpa_dim is the \textit{y}-value of the top of the columns
\item \l_tmpb_dim is the \textit{y}-value of the bottom.
\end{itemize}

\begin{itemize}
\item \l_tmpa_dim is the \textit{y}-value of the top of the columns
\item \l_tmpb_dim is the \textit{y}-value of the bottom.
\end{itemize}
Here an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,4,5,6-6}`

```
\NewDocumentCommand \@@_cellcolor { O { } m m }
{ \tl_if_blank:nF { #2 } }
{ \pgfpicture
\pgf@relevantforpicturesizefalse
\tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
\clist_map_inline:nn { #3 }
{ \@@_cut_on_hyphen:w ##1 \q_stop
\@@_qpoint:n { row - \l_tmpa_tl }
\bool_lazy_and:nnT { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
{ \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
\@@_qpoint:n { col - \l_tmpb_tl }
\int_compare:nNnTF { \l_@@_first_col_int = \l_tmpb_tl }
\pgfpathrectanglecorners
\pgfusepathqfill
\endpgfpicture
}
}
```

Here an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
\NewDocumentCommand \@@_rectanglecolor { O { } m m m }
{ \tl_if_blank:nF { #2 } }
{ \pgfpicture
\pgf@relevantforpicturesizefalse
\tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
\bool_lazy_and:nnT { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
{ \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
\@@_qpoint:n { col - \l_tmpb_tl }
\int_compare:nNnT { \l_@@_first_col_int = \l_tmpb_tl }
\pgfpathrectanglecorners
\pgfusepathqfill
\endpgfpicture
}
```

122
The command \rowcolors (accessible in the code-before) is inspired by the command \rowcolors of the package xcolor (with the option table). However, the command \rowcolors of nicematrix has not the optional argument of the command \rowcolors of xcolor. Here is an example:
\rowcolors{1}{blue!10}{respect-blocks}.

The last optional argument is for options. As of now, there is only one key available: respect-blocks.

\keys_define:nn { NiceMatrix / rowcolors }{ respect-blocks .bool_set:N = \l_@@_respect_blocks_bool , respect-blocks .default:n = true , unknown .code:n = \@@_error:n { Unknown-option-for-rowcolors } }

\NewDocumentCommand \@@_rowcolors { O { } m m m O { } }{\keys_set:nn { NiceMatrix / rowcolors } { #5 }\bool_lazy_and:nnTF \l_@@_respect_blocks_bool \{ \cs_if_exist_p:c { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } \{ \@@_rowcolors_i:nnnn { #1 } { #2 } { #3 } { #4 } \} \int_step_inline:nnn { #2 } { \int_use:N \c@iRow }{ \int_if_odd:nTF { ##1 }{ \@@_rowcolor [ #1 ] { #3 } }{ \@@_rowcolor [ #1 ] { #4 } }{ ##1 } }\int_do_until:nNnn \l_tmpa_int > \c@iRow \} \cs_new_protected:Npn \@@_rowcolors_i:nnnn { #1 } { #2 } { #3 } { #4 }{ \cs_if_exist_p:c { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } \seq_set_eq:Nc \l_tmpb_seq { \c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq \{ \@@_not_in_exterior_p:nnnn ##1 \} \int_set:Nn \l_tmpa_int { #2 } \int_do_until:nNnn \l_tmpa_int > \c@iRow \}

We don’t want to take into account a block which is completely in the “first column” of (number 0) or in the “last column”.
\seq_set_eq:Nc \l_tmb_seq { \c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } \seq_set_filter:NNn \l_tmb_seq \{ \@@_not_in_exterior_p:nnnn #1 \} \int_set:Nn \l_tmpa_int { #2 } \int_do_until:nNnn \l_tmpa_int > \c@iRow

We recall that, in the code-before, \c@iRow is the total number of rows of the array (excepted the potential exterior rows).
We compute in $\l_{tmpb_int}$ the last row covered by a block.

$$\def\seq_set_filter{\seq_filter:Nn} \def\l_{tmpb_seq}{\l_{tmpa_seq}}$$

$$\def\\_intersect\_our\_row\_p{\\_intersect\_our\_row\_p:nnnn \#1}$$

The following command return \texttt{true} when the block intersects the row $\l_{tmpa_int}$.

$$\def\cs_{new\_protected}{\cs_{new\_protected}:N\!p} \def\\_rowcolors\_ii{\\_rowcolors\_ii:nnnn \#1 \#2 \#3 \#4}$$

$$\def\\_chessboardcolors{\\_chessboardcolors:nnn m}$$

$$\def\\_cellcolor{\_cellcolor}[\#1]{\#2}$$

$$\def\\_cellcolor{\_cellcolor}[\#1]{\#3}$$

$$\def\\_not_in_exterior{\_not_in_exterior:nnnn p}$$

124
When the user uses the key \colortbl-like, the following command will be linked to \cellcolor in the tabular.

\NewDocumentCommand \@@_cellcolor_tabular { O { } m } { \tl_gput_right:Nx \g_nicematrix_code_before_tl \{ \cellcolor [ #1 ] { #2 } \{ \int_use:N \c@iRow - \int_use:N \c@jCol \} \} }

When the user uses the key rowcolor-in-tabular, the following command will be linked to \rowcolor in the tabular.

\NewDocumentCommand \@@_rowcolor_tabular { O { } m } { \tl_gput_right:Nx \g_nicematrix_code_before_tl \{ \exp_not:N \rectanglecolor [ #1 ] { #2 } \{ \int_use:N \c@iRow - \int_use:N \c@jCol \} \{ \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } \} \} }

\NewDocumentCommand \@@_columncolor_preamble { O { } m } { \int_compare:nNnT \c@iRow = 1 { \tl_gput_left:Nx \g_nicematrix_code_before_tl \{ \exp_not:N \columncolor [ #1 ] { #2 } { \int_use:N \c@jCol } \} } }

You use gput_left because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells).

\tl_gput_left:Nx \g_nicematrix_code_before_tl \{ \exp_not:N \columncolor [ #1 ] { #2 } \{ \int_use:N \c@jCol \} \} }

The vertical rules

We give to the user the possibility to define new types of columns (with \texttt{\newcolumntype} of \texttt{array}) for special vertical rules (e.g. rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command \texttt{\OnlyMainNiceMatrix} in that goal. However, that command must be no-op outside the environments of \texttt{nicematrix} (and so the user will be allowed to use the same new type of column in the environments of \texttt{nicematrix} and in the standard environments of \texttt{array}). That’s why we provide first a global definition of \texttt{\OnlyMainNiceMatrix}.

\cs_set_eq:NN \OnlyMainNiceMatrix \use:n

Another definition of \texttt{\OnlyMainNiceMatrix} will be linked to the command in the environments of \texttt{nicematrix}. Here is that definition, called \texttt{\@@_OnlyMainNiceMatrix:n}.

\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1

You use gput_left because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells).
This definition may seem complicated by we must remind that the number of row \c@iRow is incremented in the first cell of the row, after an potential vertical rule on the left side of the first cell.

The command \@@_OnlyMainNiceMatrix_i:n is only a short-cut which is used twice in the above command. This command must not be protected.

\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n \#1
\int_compare:nNnF \c@iRow = 0
\int_compare:nNnF \c@iRow = \l_@@_last_row_int { \#1 }
\}

Remember that \c@iRow is not always inferior to \l_@@_last_row_int because \l_@@_last_row_int may be equal to −2 or −1 (we can't write \int_compare:nNnT \c@iRow < \l_@@_last_row_int).

The following command will be executed in the internal-code-after. The rule will be drawn before the column \#1. \#2 is the number of consecutive occurrences of 1.

\cs_new_protected:Npn \@@_vline:nn \#1 \#2
\int_compare:nNnT { \#1 } < { \c@jCol + 2 }
\pgfpicture
\@@_vline_i:nn { \#1 } { \#2 }
\endpgfpicture
\}

\cs_new_protected:Npn \@@_vline_i:nn \#1 \#2
\tl_set:Nx \l_tmpb_tl { \#1 }
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@iRow \l_tmpa_tl
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq { \@@_test_if_vline_in_block:nnnn \#1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq { \@@_test_if_vline_in_block:nnnn \#1 }
\clist_if_empty:NF \l_@@_except_corners_clist
\@@_test_in_corner_v:
\bool_if:NTF \g_tmpa_bool
\tl_if_empty:NT \l_tmpc_tl
\tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
\tl_if_empty:NF \l_tmpc_tl
\@@_vline_ii:nnnn \l_tmpc_tl

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_tmpc_tl.

\tl_set:Nx \l_tmpb_tl \l_tmpc_tl
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@iRow \l_tmpa_tl
\}

\newcommand{\t\text{tmpa_tl}}{\l_tmpa_tl}
\newcommand{\t\text{tmpb_tl}}{\l_tmpb_tl}
\newcommand{\t\text{tmpc_tl}}{\l_tmpc_tl}

The boolean \g_tmpa_bool indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by \Block or a virtual block corresponding to a dotted line, created by \dots, \Vdots, etc.), we will set \g_tmpa_bool to false and the small vertical rule won't be drawn.

\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq { \@@_test_if_vline_in_block:nnnn \#1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq { \@@_test_if_vline_in_block:nnnn \#1 }
\clist_if_empty:NF \l_@@_except_corners_clist
\@@_test_in_corner_v:
\bool_if:NTF \g_tmpa_bool
\tl_if_empty:NT \l_tmpc_tl
\}

We keep in memory that we have a rule to draw.

\tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
\}

\tl_if_empty:NT \l_tmpc_tl
\}
\tl_if_empty:NT \l_tmpc_tl
\}

126
\cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
\begin{table}[ht]
\centering
\begin{tabular}{|c|c|}
\hline
#1 & is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the number of the rows between which the rule has to be drawn. \\
\hline
\end{tabular}
\end{table}
\dim_set:Nn \l_tmpd_dim \{ \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) \}
\pgfpathrectanglecorners
{ \pgfpoint \l_tmpb_dim \l_tmpa_dim }
{ \pgfpoint \l_tmpd_dim \l_tmpc_dim }
\pgfusepathqfill
\group_end:
\pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
\pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
\prg_replicate:nn { #2 - 1 }
{ \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
\dim_sub:Nn \l_tmpb_dim \doublerulesep
\pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
\pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
}\CT@arc@
\pgfsetlinewidth { 1.1 \arrayrulewidth }
\pgfsetrectcap
\pgfusepathqstroke
}

The following draws a complete vertical rule in the column \#1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key except-corners is not used).
\cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
{ \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

The command \@@_draw_hlines: draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as \Cdots) and in the corners (if the key except-corners is used).
\cs_new_protected:Npn \@@_draw_vlines:
{ \int_step_inline:nnn
{ \bool_if:NTF \l_@@_NiceArray_bool 1 2 } \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
{ \@@_vline:nn { ##1 } 1 }
}

The horizontal rules

The following command will be executed in the internal-code-after. The row will be drawn before the row \#1. #2 is the number of consecutive occurrences of \Hline.
\cs_new_protected:Npn \@@_hline:nn #1 #2
{ \pgfpicture
\@@_hline_i:nn { #1 } { #2 }
\endpgfpicture
\cs_new_protected:Npn \@@_hline_i:nn #1 #2
{ \l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. We, have found a column corresponding to a rule to draw, we note its number in \l_tmpc_tl.
\tl_set:Nn \l_tmpa_tl \{ #1 \}
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@jCol \l_tmpb_tl
}

\dim_set:Nn \l_tmpa_dim \{ \doublerulesep + \arrayrulewidth \}
\dim_set:Nn \l_tmpb_dim \{ \doublerulesep + \arrayrulewidth \}
\pgfpoint \l_tmpb_dim \l_tmpa_dim
\pgfpoint \l_tmpd_dim \l_tmpc_dim
\pgfsetlinewidth { 1.1 \arrayrulewidth }
\pgfsetrectcap
\pgfusepathqstroke
}

The horizontal rules

The following command will be executed in the internal-code-after. The row will be drawn before the row \#1. #2 is the number of consecutive occurrences of \Hline.
\cs_new_protected:Npn \@@_hline:nn #1 #2
{ \pgfpicture
\@@_hline_i:nn { #1 } { #2 }
\endpgfpicture
\cs_new_protected:Npn \@@_hline_i:nn #1 #2
{ \l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. We, have found a column corresponding to a rule to draw, we note its number in \l_tmpc_tl.
\tl_set:Nn \l_tmpa_tl \{ #1 \}
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@jCol \l_tmpb_tl
}

128
The boolean \texttt{\g_tmpa_bool} indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by \texttt{\Block} or a virtual block corresponding to a dotted line, created by \texttt{\Cdots}, \texttt{\Vdots}, etc.), we will set \texttt{\g_tmpa_bool} to \texttt{false} and the small horizontal rule won't be drawn.

\begin{verbatim}
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq
  { \@@_test_if_hline_in_block:nnnn ##1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq
  { \@@_test_if_hline_in_block:nnnn ##1 }
\clist_if_empty:NF \l_@@_except_corners_clist \@@_test_in_corner_h:
\bool_if:NTF \g_tmpa_bool
  { \tl_if_empty:NT \l_tmpc_tl
    { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
    { \tl_if_empty:NF \l_tmpc_tl
      \@@_hline_ii:nnnn { #1 } { #2 } \l_tmpc_tl
      { \int_eval:n { \l_tmpb_tl - 1 } }
      \tl_clear:N \l_tmpc_tl
    }
    \tl_if_empty:NF \l_tmpc_tl
    \@@_hline_ii:nnnn { #1 } { #2 } \l_tmpc_tl
    \l_tmpb_tl
    \@_hline_ii:nnnn { \int_eval:n { \l_tmpb_tl - 1 } }
    \tl_clear:N \l_tmpc_tl
  }

\cs_new_protected:Npn \@@_test_in_corner_h:
  { \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
    { \seq_if_in:NnXT \l_@@_empty_corner_cells_seq
      \l_@@_empty_corner_cells_seq
      { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
      \bool_set_false:N \g_tmpa_bool }
    { \seq_if_in:NnXT \l_@@_empty_corner_cells_seq
      \l_tmpa_tl - \l_tmpb_tl
      \int_compare:nNnTF \l_tmpa_tl = 1
      { \bool_set_false:N \g_tmpa_bool }
      { \seq_if_in:NnXT \l_@@_empty_corner_cells_seq
        \l_@@_empty_corner_cells_seq
        { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
        \bool_set_false:N \g_tmpa_bool }
    }
  }
\end{verbatim}

We keep in memory that we have a rule to draw.
\cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
\{ 
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_qpoint:n { \l_\arraycolsep + #3 }
\dim_set_eq:NN \l_tmpa_dim \pgf@x
\@@_qpoint:n { \l_\parskip + #1 }
\dim_set_eq:NN \l_tmpb_dim \pgf@y
\@@_qpoint:n { \l_\arraycolsep + #4 }
\dim_set_eq:NN \l_tmpc_dim \pgf@x
\bool_lazy_and:nnT
{ \int_compare_p:nNn { #2 } > 1 }
{ ! \tl_if_blank_p:V \CT@drsc@ }
{ \group_begin: 
\CT@drsc@
\dim_set:Nn \l_tmpd_dim
{ \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
\pgfpathrectanglecorners
\pgfpoint { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
\pgfpoint { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
\pgfusepathqfill
\group_end: }
\pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
\pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
\prg_replicate:nn { #2 - 1 }
{ \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
\dim_sub:Nn \l_tmpb_dim \doublerulesep
\pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
\pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
}\CT@arc@
\pgfsetlinewidth { 1.1 \arrayrulewidth }
\pgfsetrectcap
\pgfusepathqstroke
\}
\cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
\{ \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } \}

The command \@@_draw_hlines: draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as \Cdots and in the corners (if the key except-corners is used).

\cs_new_protected:Npn \@@_draw_hlines: 
\{ 
\int_step_inline:nnn
{ \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
{ \bool_if:NTF \l_@@_NiceArray_bool { \\succ:n \c@iRow } \c@iRow }
{ \@@_hline:nn { \#1 } \#2 } \} { \int_use:N \c@jCol } \}

The command \@@_Hline: will be linked to \Hline in the environments of nicematrix.
The argument of the command \@@_Hline_i:n is the number of successive \Hline found.

\cs_set:Npn \@@_Hline_i:n #1
{ \peek_meaning_ignore_spaces:NTF \Hline
  { \@@_Hline_ii:nn { #1 + 1 } }
  { \@@_Hline_iii:n { #1 } }
}
\cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
\cs_set:Npn \@@_Hline_iii:n #1
{ \skip_vertical:n
  { \arrayrulewidth * ( #1 ) + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } ) }
  \tl_gput_right:Nx \g_@@_internal_code_after_tl
  { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
  \ifnum 0 = `{ \fi }
}

The key hvlines

The following command tests whether the current position in the array (given by \l_tmpa_tl for the row and \l_tmpb_tl for the col) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean \l_tmpa_bool is set to false.

\cs_new_protected:Npn \@@_test_if_hline_in_block:nnnn #1 #2 #3 #4
{ \bool_lazy_all:nT
  { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
  { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
  { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
  { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
  { \bool_gset_false:N \g_tmpa_bool }
}

The same for vertical rules.

\cs_new_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4
{ \bool_lazy_all:nT
  { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
  { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
  { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
  { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
  { \bool_gset_false:N \g_tmpa_bool }
}

The key except-corners

When the key except-corners is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

\cs_new_protected:Npn \@@_compute_corners:
{
The sequence $\_\_\@empty$corner cells seq will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```
\seq_clear_new:N \_\_\@empty_corner_cells_seq
\clist_map_inline:Nn \_\_\@except_corners_clist
  \str_case:nnF { ##1 }
    \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol
    \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1
    \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 \c@jCol
    \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1
    \@@_error:nn { bad~corner } { ##1 }
  \}
```

"Computing a corner" is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence $\_\_\@empty$corner cells seq.

The six arguments of $\_\_\@compute_a_corner:nnnnnn$ are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```
\cs_new_protected:Npn \_\_\@compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
  \bool_set_false:N \l_tmpa_bool
  \int_zero_new:N \l_@@_last_empty_row_int
  \int_set:Nn \l_@@_last_empty_row_int { #1 }
  \int_step_inline:nnnn { #1 } { #3 } { #5 }
  { \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } } \bool_lazy_or:nnTF
    \cs_if_exist_p:c
    \pgf @ sh @ ns @ \_\_\@env: - ##1 - \int_eval:n { #2 } \l_tmpb_bool
    \bool_set_true:N \l_tmpa_bool
    \bool_if:NF \l_tmpa_bool
      \int_set:Nn \l_@@_last_empty_row_int { ##1 }
  }
```

For the explanations and the name of the variables, we consider that we are computing the left-upper corner.

First, we try to determine which is the last empty cell (and not in a block: we won’t add that precision any longer) in the column of number 1. The flag $\_\_\tmpa$bool will be raised when a non-empty cell is found.

```
\bool_set_false:N \_\tmpa
\int_zero_new:N \_\@last_empty_row_int
\int_set:Nn \_\@last_empty_row_int { #1 }
\int_step_inline:nnnn { #1 } { #3 } { #5 }
{ \_\@test_if_cell_in_a_block:nn { #1 } { \int_eval:n { #2 } } \bool_lazy_or:nnTF
  \cs_if_exist_p:c
  \pgf @ sh @ ns @ \_\_\@env: - #1 - \int_eval:n { #2 } }
\_\tmpb
{ \bool_set_true:N \_\tmpa
  \bool_if:NF \_\tmpa
    \int_set:Nn \_\@last_empty_row_int { #1 }
}
```

Now, you determine the last empty cell in the row of number 1.

```
\bool_set_false:N \_\tmpa
\int_zero_new:N \_\@last_empty_row_int
\int_set:Nn \_\@last_empty_row_int { #2 }
\int_step_inline:nnnn { #2 } { #4 } { #6 }
```
The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a `\diagbox`). The flag `l_tmph_bool` will be raised if the cell #1-#2 is in a block (or in a cell with a `\diagbox`).
The commands to draw dotted lines to separate columns and rows

These commands don’t use the normal nodes, the medium nor the large nodes. They only use the col nodes and the row nodes.

**Horizontal dotted lines**

The following command must *not* be protected because it’s meant to be expanded in a \noalign.

\cs_new:Npn \@@_hdottedline:
\{ \noalign { \skip_vertical:N 2\l_@@_radius_dim } \@@_hdottedline_i:
\}

On the other side, the following command should be protected.

\cs_new_protected:Npn \@@_hdottedline_i:
\{ \tl_gput_right:Nx \g_@@_internal_code_after_tl \@@_hdottedline:n \{ \int_use:N \c@iRow \} \}

The command \@@_hdottedline:n is the command written in the code-after that will actually draw the dotted line. Its argument is the number of the row before which we will draw the row.

\AtBeginDocument
\{ \tikzpicture \begin{pgfpicture} \end{pgfpicture} \}

The following command *must* be protected since it is used in the construction of \@@_hdottedline:n.

\cs_new_protected:Npn \@@_hdottedline_i:n #1
\{ \bool_set_true:N \exp_not:N \l_@@_initial_open_bool \bool_set_true:N \exp_not:N \l_@@_final_open_bool \c_@@_pgfortikzpicture_tl \@@_qpoint:n \{ \int_use:N \c@iRow \} \}

We do a translation par \l_@@_radius_dim because we want the dotted line to have exactly the same position as a vertical rule drawn by “\mid” (considering the rule having a width equal to the diameter of the dots).
The dotted line will be extended if the user uses margin (or left-margin and right-margin).
The aim is that, by standard the dotted line fits between square brackets (\hline doesn’t).
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
But, if the user uses margin, the dotted line extends to have the same width as a \hline.
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
We do a reduction by \arraycolsep for the environments with delimiters (and not for the other).
\begin{Verbatim}
\bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep - \l_@@_left_margin_dim
\end{Verbatim}
\begin{Verbatim}
\bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep + \l_@@_right_margin_dim
\end{Verbatim}
For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by 0.5 \l_@@_inter_dots_dim is \textit{ad hoc} for a better result.
\begin{Verbatim}
\tl_set:Nn \l_tmpa_tl { ( }
\tl_if_eq:NNF \l_@@_left_delim_tl \l_tmpa_tl { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
\tl_set:Nn \l_tmpa_tl { ) }
\tl_if_eq:NNF \l_@@_right_delim_tl \l_tmpa_tl { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
\end{Verbatim}
As of now, we have no option to control the style of the lines drawn by \hdottedline and the specifier “:” in the preamble. That’s why we impose the style \texttt{standard}.
\begin{Verbatim}
\tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
\@@_draw_line:
\end{Verbatim}
Vertical dotted lines
\cs_new_protected:Npn \@@_vdottedline:n #1
{ \bool_set_true:N \l_@@_initial_open_bool \bool_set_true:N \l_@@_final_open_bool
\begin{tikzpicture}
\pgfsetlinewidth{0.5pt}
\draw [dotted] (0,0) -- (\l_@@_x_initial_dim,0);
\end{tikzpicture}
\begin{pgfpicture}
\pgfsetlinewidth{0.5pt}
\draw [dotted] (0,0) -- (\l_@@_x_final_dim,0);
\end{pgfpicture}
\begin{Verbatim}
\bool_if:NTF \c_@@_tikz_loaded_bool
{ \tikzpicture
\end{tikzpicture}
\end{Verbatim}
The command \CT@arc@ is a command of colortbl which sets the color of the rules in the array. The package nicematrix uses it even if colortbl is not loaded.

\pgfrememberpicturepositiononpagetrue
\@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

We do a translation par -\l_@@_radius_dim because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).
\dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
\dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
\@@_qpoint:n { row - 1 }

We arbitrary decrease the height of the dotted line by a quantity equal to \l_@@_inter_dots_dim in order to improve the visual impact.
\dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
\@@_qpoint:n { row - \@@_succ:n \c@iRow }
\dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }

As of now, we have no option to control the style of the lines drawn by \hdottedline and the specifier “;” in the preamble. That’s why we impose the style standard.
\tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
\@@_draw_line:
}

The environment \texttt{NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.
\bool_new:N \l_@@_block_auto_columns_width_bool

As of now, there is only one option available for the environment \texttt{NiceMatrixBlock}.
\keys_define:nn { NiceMatrix / NiceMatrixBlock } { }
\auto-columns-width .code:n =
\bool_set_true:N \l_@@_block_auto_columns_width_bool
\dim_gzero_new:N \g_@@_max_cell_width_dim
\bool_set_true:N \l_@@_auto_columns_width_bool
}

\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
\int_gincr:N \g_@@_NiceMatrixBlock_int
\dim_zero:N \l_@@_columns_width_dim
\keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
At the end of the environment \{NiceMatrixBlock\}, we write in the main .aux file instructions for the column width of all the environments of the block (that’s why we have stored the number of the first environment of the block in the counter \l_@@_first_env_block_int).}

\begin{Verbatim}
\bool_if:NT \l_@@\_block_auto_columns_width_bool
\{
\cs_if_exist:cT \{ \@@_max_cell_width_ \int_use:N \g_@@\_NiceMatrixBlock_int \}
\{
\exp_args:Nc \dim_set:Nn \l_@@\_columns_width_dim
\{ \@@_max_cell_width_ \int_use:N \g_@@\_NiceMatrixBlock_int \}
\}
\}
\}
\end{Verbatim}

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

\begin{Verbatim}
\bool_if:NT \l_@@\_block_auto_columns_width_bool
\{
\iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
\{
\cs_gset:cpn { \@@_\_max_\_cell_\_width_ \int_use:N \g_@@\_NiceMatrixBlock_int }
\}
\}
\iow_shipout:Nn \@mainaux \ExplSyntaxOff
\end{Verbatim}

The extra nodes

First, two variants of the functions \dim\_min:nn and \dim\_max:nn.

\begin{Verbatim}
\cs_generate_variant:Nn \dim\_min:nn { v n }
\cs_generate_variant:Nn \dim\_max:nn { v n }
\end{Verbatim}

The following command is called in \@@\_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks dans that construction uses the standard medium nodes).

\begin{Verbatim}
\cs_new_protected:Npn \@@\_create_extra_nodes:
\{
\bool_if:nTF \l_@@\_medium_nodes_bool
\{
\bool_if:nTF \l_@@\_large_nodes_bool
\@@\_create_medium_and_large_nodes:
\@@\_create_medium_nodes:
\}
\{ \bool_if:NT \l_@@\_large_nodes_bool \@@\_create_large_nodes: \}
\}
\end{Verbatim}

We have three macros of creation of nodes: \@@\_create_medium_nodes:, \@@\_create_large_nodes: and \@@\_create_medium_and_large_nodes:.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command \@@\_computations_for_medium_nodes: to do these computations.

The command \@@\_computations_for_medium_nodes: must be used in a \{pgfpicture\}. 

137
For each row \( i \), we compute two dimensions \( l_{\text{row}_i \_\text{min}} \) and \( l_{\text{row}_i \_\text{max}} \). The dimension \( l_{\text{row}_i \_\text{min}} \) is the minimal \( y\)-value of all the cells of the row \( i \). The dimension \( l_{\text{row}_i \_\text{max}} \) is the maximal \( y\)-value of all the cells of the row \( i \).

Similarly, for each column \( j \), we compute two dimensions \( l_{\text{column}_j \_\text{min}} \) and \( l_{\text{column}_j \_\text{max}} \). The dimension \( l_{\text{column}_j \_\text{min}} \) is the minimal \( x\)-value of all the cells of the column \( j \). The dimension \( l_{\text{column}_j \_\text{max}} \) is the maximal \( x\)-value of all the cells of the column \( j \).

Since these dimensions will be computed as maximum or minimum, we initialize them to \( \text{\texttt{c\_max\_dim}} \) or \( -\text{\texttt{c\_max\_dim}} \).

```
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
{\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
  \dim_zero_new:c \l_{\text{row}_i \_\text{min}} \dim_set_eq:cN \l_{\text{row}_i \_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_zero_new:c \l_{\text{row}_i \_\text{max}} \dim_set:cn \l_{\text{row}_i \_\text{max}} {-\text{\texttt{c\_max\_dim}}}
}\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
  \dim_zero_new:c \l_{\text{column}_j \_\text{min}} \dim_set_eq:cN \l_{\text{column}_j \_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_zero_new:c \l_{\text{column}_j \_\text{max}} \dim_set:cn \l_{\text{column}_j \_\text{max}} {-\text{\texttt{c\_max\_dim}}}
}

We begin the two nested loops over the rows and the columns of the array.

```
\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
{ \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
  \dim_zero_new:c \l_{\text{row}_\_\text{min}} \dim_set_eq:cN \l_{\text{row}_\_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_zero_new:c \l_{\text{row}_\_\text{max}} \dim_set:cn \l_{\text{row}_\_\text{max}} {-\text{\texttt{c\_max\_dim}}}
\seq_if_in:NxF \g_@@_multicolumn_cells_seq \l_{\text{column}_\_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_set:cn \l_{\text{column}_\_\text{max}} {-\text{\texttt{c\_max\_dim}}}
}
```

If the cell \((i,j)\) is empty or an implicit cell (that is to say a cell after implicit ampersands &) we don’t update the dimensions we want to compute.

```
\cs_if_exist:cT \l_{\text{column}_\_\text{min}} \dim_set:cn \l_{\text{column}_\_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_set:cn \l_{\text{column}_\_\text{max}} {-\text{\texttt{c\_max\_dim}}}
```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell \((i,j)\). They will be stored in \texttt{\pgf@x} and \texttt{\pgf@y}.

```
\pgfpointanchor \l_{\text{column}_\_\text{min}} \{ \text{\texttt{south\_west}} \}
\dim_set:cn \l_{\text{row}_\_\text{min}} \\text{\texttt{c\_max\_dim}}
  \{ \dim_min:vN \l_{\text{row}_\_\text{min}} \\text{\texttt{c\_max\_dim}} \pgf@y \}
\seq_if_in:NxF \g_@@_multicolumn_cells_seq \l_{\text{row}_\_\text{min}} \\text{\texttt{c\_max\_dim}}
  \dim_set:cn \l_{\text{row}_\_\text{max}} {-\text{\texttt{c\_max\_dim}}}
\seq_if_in:NxF \g_@@_multicolumn_cells_seq \l_{\text{row}_\_\text{max}} \\text{\texttt{c\_max\_dim}}
  \pgf@x
```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell \((i,j)\). They will be stored in \texttt{\pgf@x} and \texttt{\pgf@y}.

```
\pgfpointanchor \l_{\text{column}_\_\text{max}} \{ \text{\texttt{north\_east}} \}
\dim_set:cn \l_{\text{row}_\_\text{max}} \\text{\texttt{c\_max\_dim}}
  \{ \dim_max:vN \l_{\text{row}_\_\text{max}} \\text{\texttt{c\_max\_dim}} \pgf@y \}
\seq_if_in:NxF \g_@@_multicolumn_cells_seq \l_{\text{row}_\_\text{max}} \\text{\texttt{c\_max\_dim}}
  \pgf@x
```

138
Now, we have to deal with empty rows or empty columns since we don’t have created nodes in such rows and columns.

Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

The command \@@_create_large_nodes: must be used when we want to create only the “large nodes” and not the medium ones. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first \@@_computations_for_medium_nodes: and then the command \@@_computations_for_large_nodes:

If we want to create both, we have to use \@@_create_medium_and_large_nodes:
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:

Now, we can create the “medium nodes”. We use a command \@@_create_nodes: because this command will also be used for the creation of the “large nodes”.

\tl_set:Nn \l_@@_suffix_tl { - medium }
\@@_create_nodes:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at \c@jCol (and not \g_@@_col_total_int). Idem for the rows.

\cs_new_protected:Npn \@@_computations_for_large_nodes:
\begin{verbatim}
{ \int_set:Nn \l_@@_first_row_int 1
  \int_set:Nn \l_@@_first_col_int 1
\end{verbatim}

We have to change the values of all the dimensions \l_@@_row_i_min_dim, \l_@@_row_i_max_dim, \l_@@_column_j_min_dim and \l_@@_column_j_max_dim.

\int_step_variable:nNn { \c@iRow - 1 } \@@_i:
\begin{verbatim}
{ \dim_set:cn { \l_@@_row \@@_i: _ min _ dim } { ( \dim_use:c { \l_@@_row \@@_i: _ min _ dim } + \dim_use:c { \l_@@_row \@@_succ:n \@@_i: _ max _ dim } ) / 2 } \dim_set_eq:cc { \l_@@_row \@@_succ:n \@@_i: _ max _ dim } { \l_@@_row \@@_i: _ min_dim } }
\int_step_variable:nNn { \c@jCol - 1 } \@@_j:
\begin{verbatim}
{ \dim_set:cn { \l_@@_column \@@_j: _ max _ dim } { ( \dim_use:c { \l_@@_column \@@_j: _ max _ dim } + \dim_use:c { \l_@@_column \@@_succ:n \@@_j: _ min _ dim } ) / 2 } \dim_set_eq:cc { \l_@@_column \@@_succ:n \@@_j: _ min _ dim } { \l_@@_column \@@_j: _ max_dim } }
\end{verbatim}
\end{verbatim}

Here, we have to use \dim_sub:cn because of the number 1 in the name.

\dim_sub:cn
\begin{verbatim}
{ \l_@@_column _ 1 _ min _ dim }
\l_@@_left_margin_dim
\dim_add:cn
\begin{verbatim}
{ \l_@@_column _ \int_use:N \c@jCol _ max _ dim }
\l_@@_right_margin_dim
\end{verbatim}
\end{verbatim}

The command \@@_create_nodes: is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions.
l_@@_row_i_min_dim, l_@@_row_i_max_dim, l_@@_column_j_min_dim and l_@@_column_j_max_dim. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses \l_@@_suffix_tl (medium or -large).

Now, we create the nodes for the cells of the \multicolumn. We recall that we have stored in \g_@@_multicolumn_cells_seq the list of the cells where a \multicolumn{...}{...}{...} with \textit{n} > 1 was issued and in \g_@@_multicolumn_sizes_seq the correspondent values of \textit{n}.

The command \@@_node_for_multicolumn:nn takes two arguments. The first is the position of the cell where the command \multicolumn{...}{...}{...} was issued in the format \textit{i}-\textit{j} and the second is the value of \textit{n} (the length of the “multi-cell”).
The blocks

The code deals with the command \Block. This command has no direct link with the environment \{NiceMatrixBlock\).

The options of the command \Block will be analyzed first in the cell of the array (and once again when the block will be put in the array in the code-after).

\begin{verbatim}
\keys_define:nn { NiceMatrix / Block / FirstPass }
    { l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
      l .value_forbidden:n = true ,
      r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
      r .value_forbidden:n = true ,
      c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
      c .value_forbidden:n = true ,
    }
\end{verbatim}

The following command will be linked to \Block in the environments of nicematrix. We define it with \NewDocumentCommand of xparse because it has an optional argument between < and > (for TeX instructions put before the math mode of the label). It's mandatory to use an expandable command (why?).

\begin{verbatim}
\NewExpandableDocumentCommand \@@_Block: { O { } m D < > { } m }
    { \@@_Block_i #2 \q_stop { #1 } { #3 } { #4 } }
\end{verbatim}

The first mandatory argument of \@@_Block: has a special syntax. It must be of the form \(i-j\) where \(i\) and \(j\) are the size (in rows and columns) of the block.

\begin{verbatim}
\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } { #3 } { #4 } { #5 } }
\end{verbatim}

Now, the arguments have been extracted: \#1 is \(i\) (the number of rows of the block), \#2 is \(j\) (the number of columns of the block), \#3 is the list of key-values, \#4 are the tokens to put before the math mode and \#5 is the label of the block.

\begin{verbatim}
\cs_new:Npn \@@_Block_i \#1-\#2 \q_stop { \@@_Block_ii:nnnnn { \#1 } { \#2 } { \#3 } { \#4 } { \#5 } }
\end{verbatim}

\begin{verbatim}
\keys_set_known:nn { NiceMatrix / Block / FirstPass } { \#3 }
\end{verbatim}

\begin{verbatim}
\tl_set:Nx \l_tmpa_tl { \int_eval:n { \c@iRow } \int_eval:n { \c@jCol } \int_eval:n { \c@iRow + #1 - 1 } \int_eval:n { \c@jCol + #2 - 1 } }
\seq_gput_left:NV \g_@@_pos_of_blocks_seq \l_tmpa_tl
\end{verbatim}

Now, \l_tmpa_tl contains an “object” corresponding to the position of the block whith four components, each of them surrounded by curly brackets: \{imin\}{jimin}\{imax\}{jimax}.

We store this information in the sequence \g_@@_pos_of_blocks_seq. We also store a complete description of the block in the sequence \g_@@_blocks_seq. Of course, the sequences \g_@@_pos_of_blocks_seq and \g_@@_blocks_seq are redundant, but it’s for efficiency.

In \g_@@_blocks_seq, each block is represented by an “object” with six components: \{imin\}{jimin}\{imax\}{jimax}\{options\}{contents}.

If the block is mono-column, we have a special treatment.

\begin{verbatim}
\int_compare:nNnTF { #2 } = 1
    { \int_gincr:N \g_@@_block_box_int
\end{verbatim}

142
If the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with \set@color and not \color_ensure_current: because that command seems to be bugged: it doesn’t work in XeLaTeX when fontspec is loaded.

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).
In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_pos_of_block_tl. However, maybe there were no key of the horizontal alignment and that’s why we put a key corresponding to the value of \l_@@_pos_of_block_tl, which is fixed by the type of current column.

\seq_gput_right:Nx \g_@@_blocks_seq
{ \l_tmpa_tl

In the standard case, that is to say a \Block which is not mono-column.

\seq_gput_right:Nx \g_@@_blocks_seq
{ \l_tmpa_tl

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).
The key **tikz** is for Tikz options used when the PGF node of the block is created (the “normal” block node and not the “short” one nor the “medium” one). In fact, as of now, it is *not documented*. Is it really a good idea to provide such a key?

```latex
define:n { NiceMatrix / Block / SecondPass }
{
  tikz .tl_set:N = \l_@@_tikz_tl ,
  tikz .value_required:n = true ,
  color .tl_set:N = \l_@@_color_tl ,
  color .value_required:n = true ,
  l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
  l .value_forbidden:n = true ,
  r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
  r .value_forbidden:n = true ,
  c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
  c .value_forbidden:n = true ,
  unknown .code:n = \@@_error:n { Unknown~key~for~Block }
}
```

The command \@@_draw_blocks: will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of \ialign because there may be tabulars in the \Block instructions that will be composed now.

```latex
\cs_new_protected:Npn \@@_draw_blocks:
{
  \cs_set_eq:NN \ialign \@@_old_ialign:
  \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iii:nnnnnn ##1 }
}
```

The group is for the keys.

```latex
\keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
\tl_if_empty:NF \l_@@_color_tl
{ \tl_gput_right:Nx \g_nicematrix_code_before_tl
  \exp_not:N \rectanglecolor
  { \l_@@_color_tl }
  { #1 - #2 }
  { #3 - #4 }
}
```

```latex
\cs_set_protected_nopar:Npn \diagbox #1 #2
{ \tl_gput_right:Nx \g_@@_internal_code_after_tl
  \exp_not:n { #1 } \exp_not:n { #2 } \exp_not:n { #3 } \exp_not:n { #4 }
}
```
Let's consider the following \{NiceTabular\}. Because of the instruction \!{\hspace{1cm}} in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create two nodes relative to the block: the node 1-1-block and the node 1-1-block-short. The latter will be used by nicematrix to put the label of the node. The first one won’t be used explicitly.

\begin{NiceTabular}{cc!{\hspace{1cm}}c}
  \Block{2-2}{our block} & & one \\
  & & two \\
  three & four & five \\
  six & seven & eight \\
\end{NiceTabular}

We highlight the node 1-1-block

We highlight the node 1-1-block-short

The construction of the node corresponding to the merged cells.

\begin{pgfscope}
\exp_args:Nx \pgfset { \l_@@_tikz_tl }
\@@_pgf_rect_node:nnnnn { \@@_env: - #1 - #2 - block }
\l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
\end{pgfscope}

We construct the short node.

\begin{pgfscope}
\exp_args:Nx \pgfset { \l_@@_tikz_tl }
\@@_pgf_rect_node:nnnnn { \@@_env: - #1 - #2 - block }
\l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
\end{pgfscope}

We recall that, when a cell is empty, no (normal) node is created in that cell. That's why we test the existence of the node before using it.
If all the cells of the column were empty, $l_{\text{tmpb\_dim}}$ has still the same value $c_{\text{max\_dim}}$. In that case, you use for $l_{\text{tmpb\_dim}}$ the value of the position of the vertical rule.

$$\dim_compare:nNnT \ l_{\text{tmpb\_dim}} = c_{\text{max\_dim}}$$

$\@\_qpoint:n\{\ col - #2\} \ \dim_set_eq:NN \ l_{\text{tmpd\_dim}} \ pgf@x$

$$\dim_compare:nNnT \ l_{\text{tmpd\_dim}} = - c_{\text{max\_dim}}$$

$$\int_step_inline:nnn \ l_{\@\_first\_row\_int} \ g_{\@\_row\_total\_int}$$

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function $\@\_pgf\_rect\_node:nnnn$ takes in as arguments the name of the node and two PGF points.

$$\bool_if:NT \ l_{\@\_medium\_nodes\_bool}$$

We retrieve (in $\pgf@x$) the $x$-value of the center of the block.

$$\pgfpointanchor\{\@@\_env: - ##1 - #2\} \{ \ \text{west} \}$$

$$\dim_set:Nn \ l_{\text{tmpb\_dim}} \{ \ \dim_min:nn \ l_{\text{tmpb\_dim}} \ pgf@x \}$$
{ \_\_env: - \#1 - \#2 - block - short }
{
\str_case:Vn \l_@@_pos_of_block_tl
{
  c { center }
  l { west }
  r { east }
}
}

We put the label of the block which has been composed in \l_@@_cell_box.
\pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
\pgfset { inner~sep = \c_zero_dim }
\pgfnode { rectangle }
{
\str_case:Vn \l_@@_pos_of_block_tl
{
  c { base }
  l { base-west }
  r { base-east }
}
}
{ \box_use_drop:N \l_@@_cell_box } { } { }

If the number of rows is different of 1, we will put the label of the block in using the short node (the label of the block has been composed in \l_@@_cell_box).
{
If we are in the first column, we must put the block as if it was with the key r.
\int_compare:nNnT \c@jCol = 0
{
{ \tl_set:Nn \l_@@_pos_of_block_tl r }
\int_compare:nNnT \c@jCol = \l_@@_last_col_int
{ \tl_set:Nn \l_@@_pos_of_block_tl l }
\pgftransformshift
{
\pgfpointanchor
{ \_\_env: - \#1 - \#2 - block - short }
{
\str_case:Vn \l_@@_pos_of_block_tl
{
  c { center }
  l { west }
  r { east }
}
}
\pgfset { inner~sep = \c_zero_dim }
\pgfnode { rectangle }
{
\str_case:Vn \l_@@_pos_of_block_tl
{
  c { center }
  l { west }
  r { east }
}
}
{ \box_use_drop:N \l_@@_cell_box } { } { }
\endpgfpicture
}
\group_end:
How to draw the dotted lines transparently

\cs_set_protected:Npn \@@_renew_matrix:
{\RenewDocumentEnvironment { pmatrix } { }\pNiceMatrix \endpNiceMatrix}
\RenewDocumentEnvironment { vmatrix } { }\vNiceMatrix \endvNiceMatrix
\RenewDocumentEnvironment { Vmatrix } { }\VNiceMatrix \endVNiceMatrix
\RenewDocumentEnvironment { bmatrix } { }\bNiceMatrix \endbNiceMatrix
\RenewDocumentEnvironment { Bmatrix } { }\BNiceMatrix \endBNiceMatrix}

Automatic arrays

\cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
{\int_set:Nn \l_@@_nb_rows_int { #1 }
\int_set:Nn \l_@@_nb_cols_int { #2 }
}
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
{\int_zero_new:N \l_@@_nb_rows_int\int_zero_new:N \l_@@_nb_cols_int\@@_set_size:n #4 \q_stop
\begin { NiceArrayWithDelims } { #1 } { #2 }
\int_compare:nNnT \l_@@_first_row_int = 0
{ \int_compare:nNnT \l_@@_first_col_int = 0 { & }
\prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\}
\prg_replicate:nn \l_@@_nb_rows_int
{ \int_compare:nNnT \l_@@_first_col_int = 0 { & }
\prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & #6 & } #6
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\}
\int_compare:nNnT \l_@@_last_row_int > { -2 }
{ \int_compare:nNnT \l_@@_first_col_int = 0 { & }
\prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\}
\end { NiceArrayWithDelims }
}
\cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
{\cs_set_protected:cpn { #1 AutoNiceMatrix } { #2 #3 }}

You put { } before #6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).
We define also an command \AutoNiceMatrix similar to the environment {NiceMatrix}.
\NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } } {
\group_begin:
\bool_set_true:N \l_@@_NiceArray_bool
\AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
\group_end:
}

The redefinition of the command \dotfill

First, we insert \@@_dotfill (which is the saved version of \dotfill) in case of use of \dotfill “internally” in the cell (e.g. \hbox to 1cm {\dotfill}).
\cs_set_eq:NN \@@_old_dotfill \dotfill
\cs_new_protected:Npn \@@_dotfill:\ {
\tl_gput_right:Nx \g_@@_internal_code_after_tl
{ \@@_actually_diagbox:nnnnnn
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \exp_not:n { #1 } }
{ \exp_not:n { #2 } }
}

Now, if the box if not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert \@@_dotfill (which is the saved version of \dotfill) in the cell of the array, and it will extend, since it is no longer in \l_@@_cell_box.
\cs_new_protected:Npn \@@_dotfill_iii:
{ \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

The command \diagbox

The command \diagbox will be linked to \diagbox:nn in the environments of nicematrix.
\cs_new_protected:Npn \@@_diagbox:nn #1 #2
{ \tl_gput_right:Nx \g_@@_internal_code_after_tl
\@@_actually_diagbox:nnnnnn
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \exp_not:n { #1 } }
{ \exp_not:n { #2 } }
}

We put the cell with \diagbox in the sequence \g_@@_pos_of_blocks_seq because a cell with \diagbox must be considered as non empty by the key except-corners.
The command \texttt{\diagbox} is also redefined locally when we draw a block.

The first four arguments of \texttt{\@@\_actually\_diagbox:nnnnnn} correspond to the rectangle (\textit{=}block) to slash (we recall that it’s possible to use \texttt{\diagbox} in a \texttt{\Block}). The two other are the elements to draw below and above the diagonal line.

\begin{Verbatim}
\cs_new_protected:Npn \@@\_actually\_diagbox:nnnnnn \#1 \#2 \#3 \#4 \#5 \#6
{
\pgfpicture
\pgf@relevantforpicturesizefalse
\pgfrememberpicturepositiononpagetrue
\@@\_qpoint:n \{ row - \#1 \}
\dim_set_eq:NN \l_tmpa_dim \pgf@y
\@@\_qpoint:n \{ col - \#2 \}
\dim_set_eq:NN \l_tmpb_dim \pgf@x
\pgfpathmoveto \{ \pgfpoint \l_tmpb_dim \l_tmpa_dim \}
\@@\_qpoint:n \{ row - \@@\_succ:n \{ \#3 \} \}
\dim_set_eq:NN \l_tmpc_dim \pgf@y
\@@\_qpoint:n \{ col - \@@\_succ:n \{ \#4 \} \}
\dim_set_eq:NN \l_tmpd_dim \pgf@x
\pgfpathlineto \{ \pgfpoint \l_tmpd_dim \l_tmpc_dim \}
}\pgfset{inner~sep = 1 pt}
\pgfsetroundcap
\pgfusepathqstroke
\pgfscope
\pgftransformshift \{ \pgfpoint \l_tmpb_dim \l_tmpd_dim \}
\pgfnode{rectangle}{south~west}{\@@\_math_toggle_token: \#5 \@@\_math_toggle_token: \} \} \}
\endpgfscope
\pgftransformshift \{ \pgfpoint \l_tmpb_dim \l_tmpd_dim \}
\pgfnode{rectangle}{north~east}{\@@\_math_toggle_token: \#6 \@@\_math_toggle_token: \} \} \}
\endpgfpicture
\end{Verbatim}

The command \texttt{\CT@arc@} is a command of \texttt{colortbl} which sets the color of the rules in the array. The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded.

\begin{Verbatim}
\CT@arc@
\pgfsetroundcap
\pgfusepathqstroke
\pgfscope
\pgfset { inner~sep = 1 pt }
\pgfscope
\pgftransformshift \{ \pgfpoint \l_tmpb_dim \l_tmpd_dim \}
\pgfnode { rectangle } { south~west }{\@@\_math_toggle_token: \#5 \@@\_math_toggle_token: \} \} \}
\endpgfscope
\pgftransformshift \{ \pgfpoint \l_tmpb_dim \l_tmpd_dim \}
\pgfnode { rectangle } { north~east }{\@@\_math_toggle_token: \#6 \@@\_math_toggle_token: \} \} \}
\endpgfpicture
\end{Verbatim}

The keyword \texttt{\CodeAfter}

In fact, in this subsection, we define the user command \texttt{\CodeAfter} for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment \texttt{\@@\_light\_syntax} on p. 87.

In the environments of \texttt{nicematrix}, \texttt{\CodeAfter} will be linked to the following command \texttt{\@@\_CodeAfter:}. That macro must not be protected since it begins with \texttt{\omit}.

\begin{Verbatim}
\cs_new:Npn \@@\_CodeAfter: \{ \omit \@@\_CodeAfter_i:n \}
\end{Verbatim}

However, in each cell of the environment, the command \texttt{\CodeAfter} is redefined to the following command \texttt{\@@\_CodeAfter_i:n} which do not begins with \texttt{\omit} (and thus, the user will be able to use \texttt{\CodeAfter} without error and without the need to prefix by \texttt{\omit}.

We have to catch everything until the end of the current environment (of \texttt{nicematrix}). First, we go until the next command \texttt{\end}.
We catch the argument of the command \end (in \#1).

If this is really the end of the current environment (of nicematrix), we put back the command \end and its argument in the TeX flow.

If this is not the \end we are looking for, we put those tokens in \g_nicematrix_code_after_tl and we go on searching for the next command \end with a recursive call to the command \@@_CodeAfter:n.

We process the options at package loading

We process the options when the package is loaded (with \usepackage) but we recommend to use \NiceMatrixOptions instead.

We must process these options after the definition of the environment \{NiceMatrix\} because the option renew-matrix executes the code \cs_set_eq:NN \env@matrix \NiceMatrix.

Of course, the command \NiceMatrix must be defined before such an instruction is executed.

The boolean \g_@@_footnotehyper_bool will indicate if the option footnotehyper is used.

The boolean \c_@@_footnote_bool will indicate if the option footnote is used, but quickly, it will also be set to true if the option footnotehyper is used.

The available options are (in alphabetic order):

- define-L-C-R,
- footnote,
- footnotehyper,
- renew-dots,
- renew-matrix-and-
  transparent.

The available options are (in alphabetic order):

- define-L-C-R,
- footnote,
- footnotehyper,
- renew-dots,
- renew-matrix-and-
  transparent.

The available options are (in alphabetic order):

- define-L-C-R,
- footnote,
- footnotehyper,
- renew-dots,
- renew-matrix-and-
  transparent.
The flag \c_@@_footnote_bool is raised and so, we will only have to test \c_@@_footnote_bool in order to know if we have to insert an environment \{savenotes\}.

Error messages of the package

The following command converts all the elements of a sequence (which are token lists) into strings.

\cs_new_protected:Npn \@@_convert_to_str_seq:N #1
{\seq_clear:N \l_tmpa_seq
 \seq_map_inline:Nn #1
153
The following command creates a sequence of strings (str) from a clist.
\cs_new_protected:Npn \@@_set_seq_of_str_from_clist:Nn #1 #2
{\seq_set_from_clist:Nn #1 { #2 } \@@_convert_to_str_seq:N #1}
\@@_set_seq_of_str_from_clist:Nn \c_@@_types_of_matrix_seq
{NiceMatrix, pNiceMatrix, bNiceMatrix, vNiceMatrix, BNiceMatrix, VNiceMatrix}

If the user uses too much columns, the command \@@_error_too_much_cols: is executed. This command raises an error but try to give the best information to the user in the error message. The command \seq_if_in:NVTTF is not expandable and that’s why we can’t put it in the error message itself. We have to do the test before the \@@_fatal:n.
\cs_new_protected:Npn \@@_error_too_much_cols:
{\seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
{\int_compare:nNnTF \l_@@_last_col_int = -2 { \@@_fatal:n { too~much~cols~for~matrix } }
{\bool_if:NF \l_@@_last_col_without_value_bool { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
}}}
\@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
{You~try~to~use~more~columns~than~allowed~by~your~\@@_full_name_env:.
\@@_message_hdotsfor:~The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~exterior~columns).~This~error~is~fatal.}
\@@_msg_new:nn { too~much~cols~for~array }
{You~try~to~use~more~columns~than~allowed~by~your~\@@_full_name_env:.
\@@_message_hdotsfor:~Recall~that~the~maximal~number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX:\counter~'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~This~error~is~fatal.}

For the following message, remind that the test is not done after the construction of the array but in each row. That’s why we have to put \c@jCol-1 and not \c@jCol.
\@@_msg_new:nn { too~much~cols~for~matrix }
{You~try~to~use~more~columns~than~allowed~by~your~\@@_full_name_env:.
\@@_message_hdotsfor:~Recall~that~the~maximal~number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX:\counter~'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~This~error~is~fatal.}
You try to use more columns than allowed by your \@@_full_name_env:\@@_message_hdotsfor:\ The maximal number of columns is \int_use:N \g_@@_static_num_of_col_int\ (plus the potential exterior ones). This error is fatal.

{ last-col-not-used }
{ The key 'last-col' is in force but you have not used that last column in your \@@_full_name_env:. However, you can go on. }

{ columns-not-used }
{ The preamble of your \@@_full_name_env: announces \int_use:N \g_@@_static_num_of_col_int\ columns but you use only \int_use:N \c@jCol.\ However, you can go on. }

{ in-first-col }
{ You can't use the command #1 in the first column (number 0) of the array.\ If you go on, this command will be ignored. }

{ in-last-col }
{ You can't use the command #1 in the last column (exterior) of the array.\ If you go on, this command will be ignored. }

{ in-first-row }
{ You can't use the command #1 in the first row (number 0) of the array.\ If you go on, this command will be ignored. }

{ in-last-row }
{ You can't use the command #1 in the last row (exterior) of the array.\ If you go on, this command will be ignored. }

{ bad-option-for-line-style }
{ Since you haven't loaded Tikz, the only value you can give to 'line-style' is 'standard'.\ If you go on, this option will be ignored. }

{ Unknown-option-for-xdots }
{ As for now there is only three options available here: 'color', 'line-style' and 'shorten' (and you try to use '\l_keys_key_tl').\ If you go on, this option will be ignored. }

{ Unknown-option-for-rowcolors }
{ As for now there is only one option available here: 'respect-blocks' (and you try to use '\l_keys_key_tl').\ If you go on, this option will be ignored. }

{ ampersand-in-light-syntax }
{ You can't use an ampersand (\token_to_str &) to separate columns because you have used the option 'light-syntax'.\ This error is fatal. }
You can't use \token_to_str:N \ to separate rows because you have used the option 'light-syntax'. You must use the character '\_@@_end_of_row_tl' (set-by-the-option 'end-of-row'). This error is fatal.

The key 'standard-cline' is available only in the preamble. If you go on this command will be ignored.

The value given to 'baseline'(\int_use:N \l_@@_first_row_int) is not valid. The value must be between \int_use:N \l_@@_row_total_int and \int_use:N \g_@@_row_total_int or equal to 't', 'c' or 'b'. If you go on, a value of 1 will be used.

Your @full_name_env: is empty. This error is fatal.

Your command \token_to_str:N \line{#1}{#2} in the 'code-after' can't be executed because a cell doesn't exist. If you go on this command will be ignored.

You can't use \token_to_str:N \Hdotsfor\ in an exterior column of the array. If you go on, the corresponding dotted line won't be drawn.

#1 is an incorrect specification for a corner (in-the-keys 'except-corners' and 'hlines-except-corners'). The available values are: NW, SW, NE and SE. If you go on, this specification of corner will be ignored.

In the \@_full_name_env:, you must use the option 'last-col' without value. However, you can go on for this time (the value '\l_keys_value_tl' will be ignored).

In \NiceMatrixoptions, you must use the option 'last-col' without value. However, you can go on for this time (the value '\l_keys_value_tl' will be ignored).

You try to draw a block in the cell #1-#2 of your matrix but the matrix is too small for that block. 

Your @full_name_env: is empty. This error is fatal.
The column type '#1' in your \@_full_name_env: is unknown. This error is fatal.

\@_msg_new:nn { tabularnote-forbidden } { You can't use the command \token_to_str:N \tabularnote\ -in-a-\@_full_name_env:. This command is available only in- \{NiceTabular\}, \{NiceArray\} and \{NiceMatrix\}. If you go on, this command will be ignored. }

\@_msg_new:nn { bottomule-without-booktabs } { You can't use the option 'tabular/bottomrule' because you haven't loaded 'booktabs'. If you go on, this option will be ignored. }

\@_msg_new:nn { enumitem-not-loaded } { You can't use the command \token_to_str:N \tabularnote\ -because you haven't loaded 'enumitem'. If you go on, this command will be ignored. }

\@_msg_new:nn { Wrong-last-row } { You have used 'last-row=\int_use:N \l_@@_last_row_int'-but-your- \@_full_name_env: seems to have \int_use:N \c@iRow rows. If you go on, the value of \int_use:N \c@iRow will be used for last row. You can avoid this problem by using 'last-row'=without-value (more compilations might be necessary). }

\@_msg_new:nn { Yet-in-env } { Environments of nicematrix can't be nested. This error is fatal. }

\@_msg_new:nn { Outside-math-mode } { The \@_full_name_env: can be used only in math mode-(and not in \token_to_str:N \vcenter). This error is fatal. }

\@_msg_new:nn { Bad-value-for-letter-for-dotted-lines } { The value of key '\l_keys_key_tl' must be of length 1. If you go on, it will be ignored. }

\@_msg_new:nnn { Unknown-key-for-Block } { The key '\l_keys_key_tl' is unknown for the command \Block. If you go on, it will be ignored. For a list of the available keys, type H <return>. }

\@_msg_new:nnn { Unknown-key-for-notes } { The key '\l_keys_key_tl' is unknown. If you go on, it will be ignored. For a list of the available keys about notes, type H <return>. }

157
Available options are (in alphabetic order):

\@_msg_new:nnn { Unknown key for NiceMatrixOptions }
\{ The key \l_keys_key_tl is unknown for the command \token_to_str:N \NiceMatrixOptions. If you go on, it will be ignored. For a list of the principal available keys, type H <return>. }

Available options are (in alphabetic order):

\@_msg_new:nnn { Unknown option for NiceArray }
\{ The option \l_keys_key_tl is unknown for the environment \{NiceArray\}. If you go on, it will be ignored. For a list of the principal available options, type H <return>. }

Available options are (in alphabetic order):
This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is also the options t, c and b).

\@@_msg_new:nnn { Unknown-option-for-NiceMatrix }

\{ The-option-'\l_keys_key_tl'-is-unknown-for-the-
\@@_full_name_env:. \"
If-you-go-on,-it-will-be-ignored. \"
For-a-list-of-the-*principal*-available-options,-type-H<-return>. 
\}

\{ The-available-options-are-(in-alphabetic-order):-
\b,-
\baseline,-
\c,-
\cell-space-bottom-limit,-
\cell-space-top-limit,-
\code-after,-
\code-for-first-col,-
\code-for-first-row,-
\code-for-last-col,-
\code-for-last-row,-
colorbl-like,-
columns-width,-
create-extra-nodes,-
create-medium-nodes,-
create-large-nodes,-
extra-left-margin,-
extrarightmargin,-
first-col,-
first-row,-
hlines,-
hvlines,-
hvlines-except-corners,-
last-col,-
last-row,-
left-margin,-
light-syntax,-
name,-
notes/bottomrule,-
notes/para,-
nullify-dots,-
renew-dots,-
right-margin,-
rules/color,-
rules/width,-
small,-
t,-
vlines,-
xdots/color,-
xdots/shorten-and-
xdots/line-style.
\@@_msg_new:nnn { Unknown-option-for-NiceTabular }
\{ The-option-’\l_keys_key_tl’-is-unknown-for-the-environment-
{NiceTabular}. \}
\{ If-you-go-on,-it-will-be-ignored. \}
\{ For-a-list-of-the--principal--available-options,-type-H<return>. \}
\{ The-available-options-are-(in-alphabetic-order):- b,-
baseline,-
c,-
cell-space-bottom-limit,-
cell-space-top-limit,-
code-after,-
code-for-first-col,-
code-for-first-row,-
code-for-last-col,-
code-for-last-row,-
colorbl-like,-
columns-width,-
create-extra-nodes,-
create-medium-nodes,-
create-large-nodes,-
extra-left-margin,-
extra-right-margin,-
first-col,-
first-row,-
hlines,-
hvlines,-
hvlines-except-corners,-
l,-
last-col,-
last-row,-
left-margin,-
light-syntax,-
name,-
nullify-dots,-
r,-
renew-dots,-
right-margin,-
rules/color,-
rules/width,-
small,-
t,-
vlines,-
xdots/color,-
xdots/shorten-and-
xdots/line-style. \}
18  History

The successive versions of the file nicematrix.sty provided by TeXLive are available on the SVN server of TeXLive:
https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency). Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment \texttt{NiceArray} with column types \texttt{L}, \texttt{C} and \texttt{R}.
Changes between version 1.2 and 1.3

New environment \texttt{\textbackslash pNiceArrayC} and its variants.
Correction of a bug in the definition of \texttt{\textbackslash BNiceMatrix}, \texttt{\textbackslash vNiceMatrix} and \texttt{\textbackslash VNiceMatrix} (in fact, it was a typo).
Options are now available locally in \texttt{\textbackslash pNiceMatrix} and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types \texttt{w} and \texttt{W} can now be used in the environments \texttt{\textbackslash NiceArray}, \texttt{\textbackslash pNiceArrayC} and its variants with the same meaning as in the package \texttt{array}.
New option \texttt{columns-width} to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of \texttt{nicematrix} were focused on the continuous dotted lines whereas the version 2.0 of \texttt{nicematrix} provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment \texttt{\textbackslash pNiceArrayRC}. With this new implementation, there is no restriction on the width of the columns.
The package \texttt{nicematrix} no longer loads \texttt{mathtools} but only \texttt{amsmath}.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option \texttt{code-for-first-row} is now available in the command \texttt{\textbackslash NiceMatrixOptions}.
Following a discussion on TeX StackExchange\footnote{cf. \url{tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}}, Tikz externalization is now deactivated in the environments of the package \texttt{nicematrix}.

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like \texttt{\textbackslash Cdots} issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a \texttt{\textbackslash Cdots} command (and a single \texttt{\textbackslash Vdots}).

\[
\begin{pmatrix}
C_j \\
\vdots \\
0 \\
0 \\
\end{pmatrix}
\]

Changes between version 2.1.3 and 2.1.4

Replacement of some options \texttt{\{\}} in commands and environments defined with \texttt{xparse} by \texttt{!\{\}} (because a recent version of \texttt{xparse} introduced the specifier \texttt{!} and modified the default behaviour of the last optional arguments).
See \url{www.texdev.net/2018/04/21/xparse-optimal-arguments-at-the-end}

\footnote{Before this version, there was an error when using \texttt{nicematrix} with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by \texttt{nicematrix} because they use the options \texttt{overlay} and \texttt{remember picture}.}
Changes between version 2.1.4 and 2.1.5
Compatibility with the classes revtex4-1 and revtex4-2.
Option allow-duplicate-names.

Changes between version 2.1.5 and 2.2
Possibility to draw horizontal dotted lines to separate rows with the command \hdottedline (similar to the classical command \hline and the command \hdashline of arydshln).
Possibility to draw vertical dotted lines to separate columns with the specifier “;” in the preamble (similar to the classical specifier “|” and the specifier “;” of arydshln).

Changes between version 2.2 and 2.2.1
Improvement of the vertical dotted lines drawn by the specifier “;” in the preamble.
Modification of the position of the dotted lines drawn by \hdottedline.

Changes between version 2.2.1 and 2.3
Compatibility with the column type S of siunitx.
Option hlines.

Changes between version 2.3 and 3.0
Modification of \hdotsfor. Now \hdotsfor erases the \vlines (of “|”) as \hdotsfor does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options first-row, last-row, first-col and last-col.

Changes between version 3.0 and 3.1
Command \Block to draw block matrices.
Error message when the user gives an incorrect value for last-row.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by \cudottedline, the symbol “;” (in the preamble of the array) and \line in code-after).
The starred versions of \Dots, \Dots, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by colortbl.
Correction of a bug: it was not possible to use the colon “;” in the preamble of an array when pdflatex was used with french-babel (because french-babel activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)
Option small.

Changes between version 3.2 and 3.3
The options first-row, last-row, first-col and last-col are now available in the environments \NiceMatrix, \pNiceMatrix, \bNiceMatrix, etc.
The option columns-width=auto doesn’t need any more a second compilation.
The options renew-dots, renew-matrix and transparent are now available as package options (as said in the documentation).
The previous version of nicematrix was incompatible with a recent version of expl3 (released 2019/09/30). This version is compatible.
Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange\textsuperscript{45}, optimization of Tikz externalization is disabled in the environments of nicematrix when the class standalone or the package standalone is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the code-after was not executed.

Changes between version 3.5 and 3.6

LaTeX counters \texttt{iRow} and \texttt{jCol} available in the cells of the array.

Addition of \texttt{\normalbaselines} before the construction of the array: in environments like \texttt{\{align\}} of \texttt{amsmath} the value of \texttt{\baselineskip} is changed and if the options \texttt{first-row} and \texttt{last-row} were used in an environment of \texttt{nicematrix}, the position of the delimiters was wrong.

A warning is written in the \texttt{.log} file if an obsolete environment is used.

There is no longer artificial errors \texttt{Duplicate-name} in the environments of \texttt{amsmath}.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: \texttt{code-for-first-col}, \texttt{code-for-last-col}, \texttt{code-for-first-row} and \texttt{code-for-last-row}.

New command \texttt{\pAutoNiceMatrix} and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command \texttt{\Block} when the block has only one row. With this programmation, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with \texttt{\multicolumn}.

An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands \texttt{\NiceMatrixLastEnv} and \texttt{\OnlyMainNiceMatrix}.

New options \texttt{create-medium-nodes} and \texttt{create-large-nodes}.

Changes between version 3.9 and 3.10

New option \texttt{light-syntax} (and \texttt{end-of-row}).

New option \texttt{dotted-lines-margin} for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to \texttt{first-row} and \texttt{last-row}.

\textsuperscript{45}cf. \url{tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize}
Changes between versions 3.11 and 3.12

Command \rotate in the cells of the array.
Options vlines, hlines and hvlines.
Option baseline for \texttt{NiceArray} (not for the other environments).
The name of the Tikz nodes created by the command \texttt{Block} has changed: when the command has been issued in the cell \(i-j\), the name is \(i-j\)-block and, if the creation of the “medium nodes” is required, a node \(i-j\)-block-medium is created.
If the user tries to use more columns than allowed by its environment, an error is raised by \texttt{nicematrix} (instead of a low-level error).
The package must be loaded with the option \texttt{obsolete-environments} if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command \rotate is improved when used in the “last row”.
The option \texttt{dotted-lines-margin} has been renamed in \texttt{xdots/shorten} and the options \texttt{xdots/color} and \texttt{xdots/line-style} have been added for a complete customization of the dotted lines.
In the environments without preamble (\texttt{NiceMatrix}, \texttt{pNiceMatrix}, etc.), it’s possible to use the options \texttt{l} (=L) or \texttt{r} (=R) to specify the type of the columns.
The starred versions of the commands \texttt{\textbackslash Cdots}, \texttt{\textbackslash Ldots}, \texttt{\textbackslash Vdots}, \texttt{\textbackslash Ddots} and \texttt{\textbackslash Iddots} are deprecated since the version 3.1 of \texttt{nicematrix}. Now, one should load \texttt{nicematrix} with the option \texttt{starred-commands} to avoid an error at the compilation.
The code of \texttt{nicematrix} no longer uses Tikz but only PGF. By default, Tikz is \textit{not} loaded by \texttt{nicematrix}.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on \texttt{stackoverflow}).
Better error messages when the user uses & or \textbackslash\ when \texttt{light-syntax} is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by \texttt{\textbackslash Dots}, \texttt{\textbackslash Cdots}, \texttt{\textbackslash Vdots}, \texttt{\textbackslash Ddots}, \texttt{\textbackslash Iddots}, \texttt{\textbackslash Hdotsfor} and the command \texttt{\textbackslash line} in the \texttt{code-after} with the tokens \_ and ^.
The option \texttt{baseline} is now available in all the environments of \texttt{nicematrix}. Before, it was available only in \texttt{NiceArray}.
New keyword \texttt{\textbackslash CodeAfter} (in the environments of \texttt{nicematrix}).

Changes between versions 3.15 and 4.0

New environment \texttt{NiceTabular}
Commands to color cells, row and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys \texttt{cell-space-top-limit} and \texttt{cell-space-bottom-limit}
New command \texttt{\textbackslash diagbox}
The key \texttt{hvline} don’t draw rules in the blocks (commands \texttt{\textbackslash Block}) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write \texttt{\textbackslash begin\{}\texttt{pNiceMatrix}a&b\textbackslash end\{}\texttt{pNiceMatrix}\textsuperscript{-2} with the expected result.
Changes between versions 4.2 and 4.3

The horizontal centering of the content of a \Block is correct even when an instruction such as \( \texttt{\vquad} \) is used in the preamble of the array.
It’s now possible to use the command \Block in the “last row”.

Changes between versions 4.3 and 4.4

New key \texttt{hvlines-except-corners}.

Changes between versions 4.4 and 5.0

Use of the standard column types \texttt{l, c, r} instead of \texttt{L, C, R}.
It’s now possible to use the command \texttt{\diagbox} in a \Block.
Command \texttt{\tabularnote}

Changes between versions 5.0 and 5.1

The vertical rules specified by | in the preamble are not broken by \texttt{\hline} (and other).
Environment \{\texttt{NiceTabular\*}\}
Command \texttt{\vdotsfor} similar to \texttt{\Hdotsfor}
The variable \texttt{\g_nicematrix\_code\_after\_tl} is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by | or || in the preamble respect the blocks.
Key \texttt{\respect\_blocks} for \texttt{\rowcolors} (with a s) in the code-before.
The variable \texttt{\g_nicematrix\_code\_before\_tl} is now public.
The key \texttt{\baseline} can take in as value of the form \texttt{line-i} to align the \texttt{\hline} in the row \texttt{i}.
The key \texttt{hvlines-except-corners} may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys \texttt{l, r} and \texttt{1} for the command \Block.
It’s possible to use the key \texttt{\draw\_first} with \texttt{\Ddots} and \texttt{\Iddots} to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key \texttt{\tabularnote}.
Different behaviour for the mono-column blocks.

Changes between versions 5.3 and 5.4

The user must never put \texttt{\omit} before \texttt{\Code\_After}.
Correction of a bug: the tabular notes \texttt{\tabularnotes} were not composed when present in a block (except a mono-column block).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>\texttt{@_Block_i}</th>
<th>4244, 4245</th>
</tr>
</thead>
<tbody>
<tr>
<td>@commands: \texttt{@_Block}</td>
<td>\texttt{@_Block}</td>
<td>1046, 4243</td>
</tr>
</tbody>
</table>
\pgfscope
\pgfset {365, 390, 864, 4438, 4509, 4540, 4679}
\pgfsetbaseline {862}
\pgfsetlinewidth {3647, 3764}
\pgfsetrectcap {3648, 3765}
\pgfsetroundcap {4676}
\pgfsyspdfmark {209, 210}
\pgftransformshift {2757}
\pgftransformrotate {2757}
\pgfsetfxfill {371, 396, 2751, 4508, 4527, 4681, 4685}
\pgfsetfxfillstroke {3649, 3766, 4677}
\phantom {299, 2924, 2939, 2962, 2985}
\pic {53}
\providedocumentcommand {4560}
\prg commands:
\prg_do nothing {170, 179, 185, 213, 465, 1007, 2204}
\prg_new conditional:Nn {3464, 3472}
\prg_replicate:nn {344, 1875, 3073, 3639, 3756, 4509, 4593, 4696, 4602}
\prg_return false {3469, 3481}
\prg_return true {3470, 3480}
\processkeys {4736}
\providedocumentcommand {53}
\providesexplpackage {4}
\quad {325}
\quad {114, 115, 131, 132, 153, 155, 1113, 1357, 1416, 1748, 1751, 3208, 3222, 3223, 3263, 3285, 3286, 3319, 3320, 3349, 3380, 3391, 4213, 4220, 4244, 4245, 4575, 4584}
\quad {1158, 2210, 3505, 4404}
\quad {353}
\quad {206}
\quad {1362}
\quad {147, 148}
\quad {189}
\quad {4559, 4562, 4565, 4568, 4571}
\quad {1, 3, 9, 10, 11}
\quad {1322, 1706, 1718}
\quad {1047}
\quad {981, 1150}
\quad {1160}
\quad {1073}
\quad {253, 254}
\quad {342, 347}
\quad {253}
\quad {693}
\quad {3010, 4501, 4513, 4532, 4544}
\quad {1281, 1401, 1568, 3823}
\quad {1563, 1667}
\quad {2222}
\quad {1082, 2021, 2030, 2059, 2068, 2076, 4611}
\quad {789, 875, 935, 1081, 1182, 1200, 1795, 1820, 1838, 1869, 1895, 1918, 2020, 2029, 2097, 2118, 4200, 4227}
\quad {78, 252, 915, 1419, 1447, 1524, 1544, 1645, 1778, 4698}
\quad {454}
\quad {1563, 1666}
\quad {248, 425, 437, 692}
\quad {1560, 1664}
Contents

1 The environments of this package ........................................ 2
2 The vertical space between the rows .................................. 2
3 The vertical position of the arrays .................................... 3
4 The blocks ........................................................................... 4
  4.1 General case ................................................................... 4
  4.2 The mono-column blocks .................................................. 5
  4.3 A small remark .............................................................. 5
5 The rules .............................................................................. 6
  5.1 Some differences with the classical environments ................. 6
    5.1.1 The vertical rules ...................................................... 6
    5.1.2 The command \vline ................................................. 6
  5.2 The thickness and the color of the rules ............................... 7
  5.3 The keys hlines and vlines .............................................. 7
  5.4 The key hvlines ................................................................ 7
  5.5 The key hvlines-except-corners ....................................... 8
  5.6 The command \diagbox ...................................................... 8
  5.7 Dotted rules .................................................................... 9
6 The color of the rows and columns ...................................... 9
  6.1 Use of colortbl .............................................................. 9
  6.2 The tools of nicematrix in the code-before ......................... 10
  6.3 Color tools with the syntax of colortbl .............................. 12
7 The width of the columns ................................................... 13
8 The exterior rows and columns ............................................ 14
<table>
<thead>
<tr>
<th>Section Number</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>The continuous dotted lines</td>
<td>16</td>
</tr>
<tr>
<td>9.1</td>
<td>The option nullify-dots</td>
<td>17</td>
</tr>
<tr>
<td>9.2</td>
<td>The commands \Hdotsfor and \Vdotsfor</td>
<td>17</td>
</tr>
<tr>
<td>9.3</td>
<td>How to generate the continuous dotted lines transparently</td>
<td>18</td>
</tr>
<tr>
<td>9.4</td>
<td>The labels of the dotted lines</td>
<td>19</td>
</tr>
<tr>
<td>9.5</td>
<td>Customization of the dotted lines</td>
<td>19</td>
</tr>
<tr>
<td>9.6</td>
<td>The dotted lines and the rules</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>The code-after</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>The notes in the tabulars</td>
<td>21</td>
</tr>
<tr>
<td>11.1</td>
<td>The footnotes</td>
<td>21</td>
</tr>
<tr>
<td>11.2</td>
<td>The notes of tabular</td>
<td>21</td>
</tr>
<tr>
<td>11.3</td>
<td>Customisation of the tabular notes</td>
<td>23</td>
</tr>
<tr>
<td>11.4</td>
<td>Use of {NiceTabular} with threeparttable</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Other features</td>
<td>25</td>
</tr>
<tr>
<td>12.1</td>
<td>Use of the column type S of siunitx</td>
<td>25</td>
</tr>
<tr>
<td>12.2</td>
<td>Alignment option in {NiceMatrix}</td>
<td>25</td>
</tr>
<tr>
<td>12.3</td>
<td>The command \rotate</td>
<td>25</td>
</tr>
<tr>
<td>12.4</td>
<td>The option small</td>
<td>26</td>
</tr>
<tr>
<td>12.5</td>
<td>The counters iRow and jCol</td>
<td>26</td>
</tr>
<tr>
<td>12.6</td>
<td>The option light-syntax</td>
<td>27</td>
</tr>
<tr>
<td>12.7</td>
<td>The environment {NiceArrayWithDelims}</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>Use of Tikz with nicematrix</td>
<td>28</td>
</tr>
<tr>
<td>13.1</td>
<td>The nodes corresponding to the contents of the cells</td>
<td>28</td>
</tr>
<tr>
<td>13.2</td>
<td>The “medium nodes” and the “large nodes”</td>
<td>29</td>
</tr>
<tr>
<td>13.3</td>
<td>The “row-nodes” and the “col-nodes”</td>
<td>30</td>
</tr>
<tr>
<td>14</td>
<td>API for the developers</td>
<td>31</td>
</tr>
<tr>
<td>15</td>
<td>Technical remarks</td>
<td>32</td>
</tr>
<tr>
<td>15.1</td>
<td>Definition of new column types</td>
<td>32</td>
</tr>
<tr>
<td>15.2</td>
<td>Diagonal lines</td>
<td>32</td>
</tr>
<tr>
<td>15.3</td>
<td>The “empty” cells</td>
<td>33</td>
</tr>
<tr>
<td>15.4</td>
<td>The option exterior-arraycolsep</td>
<td>33</td>
</tr>
<tr>
<td>15.5</td>
<td>Incompatibilities</td>
<td>33</td>
</tr>
<tr>
<td>16</td>
<td>Examples</td>
<td>34</td>
</tr>
<tr>
<td>16.1</td>
<td>Notes in the tabulars</td>
<td>34</td>
</tr>
<tr>
<td>16.2</td>
<td>Dotted lines</td>
<td>35</td>
</tr>
<tr>
<td>16.3</td>
<td>Dotted lines which are no longer dotted</td>
<td>37</td>
</tr>
<tr>
<td>16.4</td>
<td>Width of the columns</td>
<td>37</td>
</tr>
<tr>
<td>16.5</td>
<td>How to highlight cells of the matrix</td>
<td>38</td>
</tr>
<tr>
<td>16.6</td>
<td>Direct use of the Tikz nodes</td>
<td>41</td>
</tr>
<tr>
<td>17</td>
<td>Implementation</td>
<td>42</td>
</tr>
<tr>
<td>18</td>
<td>History</td>
<td>161</td>
</tr>
</tbody>
</table>

Index: 166