The package \texttt{nicematrix}\footnote{This document corresponds to the version 5.3 of \texttt{nicematrix}, at the date of 2020/09/03.}

F. Pantigny
fpantigny@wanadoo.fr
September 3, 2020

Abstract

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

\begin{tabular}{|l|l|l|l|}
\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}

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

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

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

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

Most features of \texttt{nicematrix} may be used without explicit use of \texttt{PGF} or \texttt{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).

\textbf{Important}

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

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

\texttt{\usepackage\{define-L-C-R\}\{nicematrix\}}
1 The environments of this package

The package \texttt{nicematrix} defines the following new environments.

\begin{verbatim}
\{NiceTabular\}  \{NiceArray\}  \{NiceMatrix\}
\{NiceTabular*\} \{pNiceArray\} \{pNiceMatrix\}
\{bNiceArray\} \{bNiceMatrix\}
\{vNiceArray\} \{vNiceMatrix\}
\{bNiceMatrix\}
\end{verbatim}

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 (\texttt{)}} 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}.

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

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{verbatim}
$\begin{pmatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14 
\end{pmatrix}$(1.2)
\end{verbatim}

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

\begin{verbatim}
$\begin{pNiceMatrix}$
\frac12 & -\frac12 \\
\frac13 & \frac14 
\end{pNiceMatrix}$(1.2)
\end{verbatim}

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

$\begin{pNiceMatrix}$
\frac12 & -\frac12 \\
\frac13 & \frac14 
\end{pNiceMatrix}$

\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}.}
3 The vertical position of the arrays

The package {nicematrix} provides an option \texttt{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.

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

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

In the following example, we use the option \texttt{t} (equivalent to \texttt{baseline=t}) immediately after an \texttt{item} of list. One should remark that the presence of a \texttt{\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 \texttt{array}, one must use \texttt{\firsthline}.

\begin{enumerate}
\item an item
\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 \texttt{booktabs}: \texttt{\toprule}, \texttt{\bottomrule}, \texttt{\midrule}, etc.

\begin{enumerate}
\item an item
\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}

New 5.2 It’s also possible to use the key \texttt{baseline} to align a matrix on an horizontal rule (drawn by \texttt{\hline}). In this aim, one should give the value \texttt{line-i} where \texttt{i} is the number of the row following the horizontal rule.

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

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

In the environments of \nicematx, 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 don’t create space by itself.

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.

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

One should remark that the horizontal centering of the contents of the blocks is correct even when an instruction such as \texttt{!\{\quad\}} 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!{\quad}ccc!{\quad}ccc@{}}
\toprule
& \Block{1-3}{\textbf{First group}} & & & \Block{1-3}{\textbf{Second group}} & & \\
\midrule
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}

\[\textbf{New 5.3}\]

It’s possible to use with the command \Block the options l, r and c for the horizontal positionning.

It’s also possible to use the command \Block in mathematical matrices.
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 \texttt{\large}, \texttt{\Large} and \texttt{\LARGE}. That’s why the command \texttt{\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}
\texttt{\Block{3-3}}<\texttt{\Large}>
A & \vphantom{\HRule} & \vphantom{\HRule} & 0 \\
& \vphantom{\HRule} & \vphantom{\HRule} & \vdots \\
& \vphantom{\HRule} & \vphantom{\HRule} & 0 \\
\hline
0 & \texttt{\HRule} & 0 & 0
\end{bNiceArray}

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

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

New 5.2 However, the vertical rules are not drawn in the blocks.

If you use \texttt{booktabs} (which provides \texttt{\toprule}, \texttt{\midrule}, \texttt{\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}{|ccc|c|}
\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, I) to draw vertical rules with the standard behaviour of \texttt{array}.

\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}.}
However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 32):

\newcolumntype{I}{!{\OnlyMainNiceMatrix{\vrule}}}

5.1.2 The command \cline

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

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

\setlength{\arrayrulewidth}{2pt}
\begin{tabular}{cccc} \hline
A&B&C&D \\
\cline{2-2}
A&B&C&D \\
\end{tabular}

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

\setlength{\arrayrulewidth}{2pt}
\begin{NiceTabular}{cccc} \hline
A&B&C&D \\
\cline{2-2}
A&B&C&D \\
\end{NiceTabular}

5.2 The thickness and the color of the rules

The environments of \nicematrix provide a key 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 \arrayrulewidth.

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

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

\begin{NiceTabular}{|ccc|}[rules/color=gray!0.9,rules/width=1pt] \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 `\OnlyMainNiceMatrix` described on page 32.

5.3 The keys hlines and vlines

The key hlines draws all the horizontal rules and the key 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 \{pNiceMatrix\} or \{bNiceArray\}) the exteriors rules are not drawn (as expected).

\$\begin{pNiceMatrix}[vlines,rules/width=0.2pt] 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \end{pNiceMatrix}\$

6
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).

\setlength{\arrayrulewidth}{1pt}
\begin{NiceTabular}{cccc}[hvlines, rules/color=blue]
rose & tulipe & marguerite & dahlia \\
violette & \Block{2-2}{\LARGE\textcolor{blue}{fleurs}} & & souci \\
pervenche & & & lys \\
arum & iris & jacinthe & muguet
\end{NiceTabular}

5.5 The key 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 & A \\
& \Block{2-2}{B} & & A & A \\
& & & & A \\
\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.

\textbf{New 5.2} 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 NW, SW, NE and SE (\textit{north west, south west, north east and south east}).
\begin{NiceTabular}{*{6}{c}}%   [hvlines-except-corners=NE,cell-space-top-limit=3pt]  
\hline & & & & & & \hline  
\hline 1\hline 1&1\hline 1&2&1\hline 1&3&3&1\hline 1&4&6&4&1\hline 1&5&10&10&5&1
\end{NiceTabular}

5.6 The command \texttt{\textbackslash diagbox}

The command \texttt{\textbackslash diagbox} (inspired by the package \texttt{diagbox}), allows, when it is used in a cell, to slash that cell diagonally downwards.\footnote{The author of this document considers that type of construction as graphically poor.}

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

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

5.7 Dotted rules

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

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

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

\begin{NiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{NiceMatrix}

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

Remark: In the package \texttt{array} (on which the package \texttt{nicematrix} relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule\footnote{In fact, this is true only for \texttt{\textbackslash hline} and “|” but not for \texttt{\textbackslash clines}: cf p. 6.}. In \texttt{nicematrix}, the dotted lines drawn by \texttt{\textbackslash 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 is 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 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.

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 \texttt{code-before} for some code that will be executed before the drawing of the tabular. In this \texttt{code-before}, new commands are available: \texttt{\cellcolor}, \texttt{\rectanglecolor}, \texttt{\rowcolor}, \texttt{\columncolor}, \texttt{\rowcolors} and \texttt{\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 \texttt{\cellcolor} takes its name from the command \texttt{\cellcolor} of colortbl.

This command takes in as mandatory arguments a color and a list of cells, each of which with the format \texttt{i-j} where \texttt{i} is the number of row and \texttt{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
\end{NiceTabular}

A command \texttt{\cellcolor} generates only one instruction \texttt{fill} (coded \texttt{f}) in the resulting PDF.
• The command \texttt{\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 \texttt{\rowcolor} takes its name from the command \texttt{\rowcolor} of \texttt{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-} 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-}]
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5 \\
a_6 & b_6 & c_6 \\
a_7 & b_7 & c_7 \\
a_8 & b_8 & c_8 \\
a_9 & b_9 & c_9 \\
a_{10} & b_{10} & c_{10} \\
\end{NiceArray}\$

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

• The command \texttt{\columncolor} takes its name from the command \texttt{\columncolor} of \texttt{colortbl}. Its syntax is similar to the syntax of \texttt{\rowcolor}.

• The command \texttt{\rowcolors} (with a \texttt{s}) takes its name from the command \texttt{\rowcolors} of \texttt{xcolor}.

The \texttt{s} emphasizes the fact that there is \textit{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 \\
Stephen & 8 \\
Sarah & 18 \\
Ashley & 20 \\
Henry & 14 \\
Madison & 15 \\
\end{NiceTabular}

\begin{NiceTabular}{|l|}
\hline
John & 12 \\
Stephen & 8 \\
Sarah & 18 \\
Ashley & 20 \\
Henry & 14 \\
Madison & 15 \\
\end{NiceTabular}

**New 5.2** There is a key respect-blocks for the instruction \texttt{\rowcolors}. With that key, the “rows” alternately colored may extend over several rows if they have to incorporate blocks.

\footnote{The command \texttt{\rowcolors} of \texttt{xcolor} is available when \texttt{xcolor} is loaded with the option \texttt{table}.}
The command \texttt{\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}

\begin{NiceMatrix}
\begin{NiceTabular}
\rowcolors{1}{blue!10}{[respect-blocks]}
\Block{l}{2-1}{John} & 12 \\
& 13 \\
Stephen & 8 \\
\Block{l}{3-1}{Sarah} & 18 \\
& 17 \\
& 15 \\
Ashley & 20 \\
Henry & 14 \\
\Block{l}{2-1}{Madison} & 15 \\
& 19
\end{NiceTabular}
\end{NiceMatrix}

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

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

\begin{NiceMatrix}
\begin{NiceTabular}
\rowcolors{1}{red!15}{1-2} \rowcolors{3}{blue!15}{}
\toprule
\Block{2-1}{Product} & \Block{1-3}{dimensions (cm)} & \ & \ & \Block{2-1}{\rotate Price} \\
\cmidrule(rl){2-4}
& 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}
\end{NiceMatrix}

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}):

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

\NewDocumentCommand { \Blue } { } { \columncolor{blue!15} }
\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 \cellcolor, \rowcolor or \columncolor will generate an instruction fill (coded f) in the resulting PDF. In cases of juxtaposed colored rectangles, one may have a thin white color line in some PDF viewers. 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}[colortbl-like]{ccc}
\[\text{code-before = \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{tabular}{|c|c|c|}
\hline
Last name & First name & Birth day \\
\hline
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\hline
\end{tabular}

\footnote{For example SumatraPDF, which uses MuPDF of Artifex Software, or PDF.js used by Firefox.}

12
7 The width of the columns

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

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

\begin{pNiceMatrix}[columns-width = 1cm]
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{pNiceMatrix}

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 \texttt{columns-width} option: 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{pNiceMatrix}[columns-width = auto]
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{pNiceMatrix}

Without surprise, it’s possible to fix the minimal width of the columns of all the matrices of a current scope with the command \texttt{\NiceMatrixOptions}.

\begin{NiceMatrixOptions}[columns-width=10mm]
\end{NiceMatrixOptions}

\begin{pNiceMatrix}
a & b \\
c & d
\end{pNiceMatrix}

\begin{pNiceMatrix}
a & b \\
c & d
\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 \texttt{auto-columns-width}.\footnote{At this time, this is the only usage of the environment \{NiceMatrixBlock\} but it may have other usages in the future.} The environment \{NiceMatrixBlock\} has no direct link with the command \texttt{\Block} presented previously in this document (cf. p. 4).

\begin{NiceMatrixBlock}[auto-columns-width]
\begin{bNiceMatrix}
9 & 17 \\
-2 & 5
\end{bNiceMatrix}

\begin{bNiceMatrix}
1 & 1245345 \\
345 & 2
\end{bNiceMatrix}
\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]
\begin{pNiceMatrix}[first-row, last-row, first-col, last-col, nullify-dots]
\begin{pmatrix}
\end{pmatrix}
\end{pNiceMatrix}
\end{pNiceMatrix}

The dotted lines have been drawn with the tools presented p. 15.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. \texttt{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 r for the first column and l for the last one.

- One may wonder how 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 \texttt{NiceTabular} and \texttt{pNiceArray}, the number of columns can obviously be computed from the preamble.
  - When the option light-syntax (cf. p. 27) is used, 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 light-syntax). The analysis of this whole body gives the number of rows (but not the number of columns).
  - In the other cases, nicematrix compute the number of rows and columns during the first compilation and write the result in the aux file for the next run.

However, it’s possible to provide the number of the last row and the number of the last column as values of the options last-row and last-col, tending to an acceleration of the whole compilation of the document. That’s what we will do throughout the rest of the document.

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

\[
\begin{array}{cccc}
L_1 & a_{11} & a_{12} & a_{13} & a_{14} \\
\vdots & a_{21} & a_{22} & a_{23} & a_{24} \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} \\
& C_1 & \Cdots & & C_4
\end{array}
\]

Remarks

- As shown in the previous example, the horizontal and vertical rules don’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 `\OnlyMainNiceMatrix` described on page 32.

- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to `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`.

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 on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it’s an horizontal line; for `\Vdots`, it’s a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It’s possible

\begin{itemize}
  \item 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`.
  \item The precise definition of a “non-empty cell” is given below (cf. p. 33).
\end{itemize}
to change the color of these lines with the option color.\textsuperscript{12}

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

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

\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:

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

In the first column of this exemple, 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:

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command \ for the vertical dimension and a command \hspace in a cell for the horizontal dimension.\textsuperscript{13}

However, a command \hspace might interfer 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 \\
 & & & \Vdots \\
 0 & \Cdots & \hspace*{1cm} & 0
\end{bNiceMatrix}

\textsuperscript{12}It’s also possible to change the color of all theses dotted lines with the option xdots/color (xdots to remind that it works for \Cdots, \Ldots, \Vdots, etc.): cf. p. 19.

\textsuperscript{13}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
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.

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

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).

\[ D = \begin{pNiceMatrix} \nullify-dots \\
\nullify-dots \\
\nullify-dots \end{pNiceMatrix} \]

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 \Dots, \Ldots, \Vdots, etc.), the dotted line drawn with \Hdotsfor extends until the contents of the cells on both sides.

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

However, if these cells are empty, the dotted line extends only in the cells specified by the argument of \Hdotsfor (by design).

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

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] & \hspace*{20mm} & C[a_1,a_1^{(p)}] & \Cdots & C[a_1,a_n^{(p)}] \\
Vdots & \Ddots & Vdots & & \Hdotsfor{1} & \Vdots & \Ddots & Vdots \\
C[a_n,a_1] & \Cdots & C[a_n,a_n] & & C[a_n,a_1^{(p)}] & \Cdots & C[a_n,a_n^{(p)}] \\
\rule{0pt}{15mm} & \Vdotsfor{1} & & \Ddots & & \Vdotsfor{1} \\
C[a_1^{(p)},a_1] & \Cdots & C[a_1^{(p)},a_n] & & C[a_1^{(p)},a_1^{(p)}] & \Cdots & C[a_1^{(p)},a_n^{(p)}] \\
Vdots & \Ddots & Vdots & & \Hdotsfor{1} & \Vdots & \Ddots & Vdots \\
C[a_n^{(p)},a_1] & \Cdots & C[a_n^{(p)},a_n] & & C[a_n^{(p)},a_1^{(p)}] & \Cdots & C[a_n^{(p)},a_n^{(p)}]
\end{bNiceMatrix}

\[ \begin{bmatrix}
C[a_1,a_1] & \cdots & C[a_1,a_n] \\
C[a_n,a_1] & \cdots & C[a_n,a_n] \\
C[a_1^{(p)},a_1] & \cdots & C[a_1^{(p)},a_n] \\
C[a_n^{(p)},a_1] & \cdots & C[a_n^{(p)},a_n]
\end{bmatrix}
\]

\[
\begin{bmatrix}
C[a_1,a_1^{(p)}] & \cdots & C[a_1,a_n^{(p)}] \\
C[a_n,a_1^{(p)}] & \cdots & C[a_n,a_n^{(p)}] \\
C[a_1^{(p)},a_1^{(p)}] & \cdots & C[a_1^{(p)},a_n^{(p)}] \\
C[a_n^{(p)},a_1^{(p)}] & \cdots & C[a_n^{(p)},a_n^{(p)}]
\end{bmatrix}
\]

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 `renew-dots` and `renew-matrix`.\(^{14}\)

- **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 `\dots` (“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

\(^{14}\)The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.
\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. 20) 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.

\begin{bNiceMatrix}
1 & \hspace*{1cm} & 0 \\
& \Ddots^{n \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. 20) 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 cm (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.}
In order to provide lines with rounded dots in the style of those provided by `\dots` (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 Tikz pathes (with the exception of “color”, “shorten >” and “shorten <”).

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

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

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

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

\section{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. 9.}

A special command, called `\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}[xdots/shorten = 0.6 em]
\begin{code-after}=\line{2-2}{3-3}\end{code-after}
I & 0 & \Cdots & 0 \\
0 & I & \Ddots & \Vdots \\
\Vdots & \Ddots & I & 0 \\
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{\texttt{Code-After}} at the end of the environment, after the keyword \texttt{CodeAfter}\footnote{In some circumstances, one must put \texttt{\texttt{Code-After}}. \texttt{\texttt{Code-After}} is a keyword of TeX which cancels the pattern of the current cell.}. For an example, cf. p. 38.

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) and 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{\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 \tabularnote{\begin{enumerate}
  \item Achard is an old family of the Poitou.
  \item The name Lefebvre is an alteration of the name Lefebure.
\end{enumerate}}
& Jacques & June 5, 2005 \tabularnote{\begin{enumerate}
  \item Achard is an old family of the Poitou.
  \item The name Lefebvre is an alteration of the name Lefebure.
\end{enumerate}}
& Mathilde & January 23, 1975 \tabularnote{\begin{enumerate}
  \item Achard is an old family of the Poitou.
  \item The name Lefebvre is an alteration of the name Lefebure.
\end{enumerate}}
Vanesse & Stephany & October 30, 1994 \tabularnote{\begin{enumerate}
  \item Achard is an old family of the Poitou.
  \item The name Lefebvre is an alteration of the name Lefebure.
\end{enumerate}}
Dupont & Chantal & January 15, 1998 \tabularnote{\begin{enumerate}
  \item Achard is an old family of the Poitou.
  \item The name Lefebvre is an alteration of the name Lefebure.
\end{enumerate}}
\midrule
\bottomrule
\end{NiceTabular}
Table 1: Use of \texttt{\textbackslash tabularnote}\textsuperscript{a}

<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\textsuperscript{b,c}</td>
<td>Florence</td>
<td>90</td>
</tr>
<tr>
<td>Schoelcher</td>
<td>Victor</td>
<td>89\textsuperscript{d}</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>

\textsuperscript{a} It’s possible to put a note in the caption.
\textsuperscript{b} Considered as the first nurse of history.
\textsuperscript{c} Nicknamed “the Lady with the Lamp”.
\textsuperscript{d} The label of the note is overlapping.

- If you have several successive commands \texttt{\textbackslash tabularnote}{...} with no space at all between them, the labels of the corresponding notes are composed together, separated by commas (this is similar to the option \texttt{multiple} of \texttt{footmisc} for the footnotes).

- If a command \texttt{\textbackslash tabularnote}{...} is exactly at the end of a cell (with no space at all after), the label of the note is composed in an overlapping position (towards the right). This structure may provide a better alignment of the cells of a given column.

- If the key \texttt{notes/para} is used, the notes are composed at the end of the array in a single paragraph (as with the key \texttt{para} of \texttt{threeparttable}).

- If the package \texttt{booktabs} has been loaded (before or after \texttt{nicematrix}), the key \texttt{notes/bottomrule} draws a \texttt{\bottomrule} of \texttt{booktabs} after the notes.

- The command \texttt{\textbackslash tabularnote} may be used before the environment of \texttt{nicematrix}. Thus, it’s possible to use it on the title inserted by \texttt{\caption} in an environment \texttt{\{table\}} of \LaTeX.

- It’s possible to create a reference to a tabular note created by \texttt{\textbackslash tabularnote} (with the usual command \texttt{\label} used after the \texttt{\textbackslash tabularnote}).

For an illustration of some of those remarks, see table 1, p. 22. This table has been composed with the following code.

\begin{table}
\setlength{\belowcaptionskip}{1ex}
\centering
\caption{Use of \texttt{\textbackslash tabularnote}\tabularnote{It’s possible to put a note in the caption.}}
\label{t:tabularnote}
\begin{NiceTabular}{@{}llc@{}}
\toprule
Last name & First name & Length of life \\
\midrule
Churchill & Wiston & 91 \\
Nightingale\tabularnote{Considered as the first nurse of history.}\tabularnote{Nicknamed “the Lady with the Lamp”} & Florence & 90 \\
Schoelcher & Victor & 89 \\
Touchet & Marie & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}
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 achieved by the following code:

  \texttt{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 achieved by:

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

  The command \texttt{\nobreak} is for the event that the option \texttt{para} is used.

• The notes are composed at the end of the tabular by using internally a style of list of \texttt{enumitem}.

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

  Initial value: noitemsep , leftrmargin = * , 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 \texttt{booktabs} (see for example the table 1, p. 22).

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

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

• The key `notes/code-before` est une token list inserted by \texttt{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 \texttt{\footnotesize}, he should use that key:

  \texttt{NiceMatrixOptions\{notes/code-before = \footnotesize \color{gray}}}\}

  It’s also possible to add \texttt{\raggedright} or \texttt{\RaggedRight} in that key (\texttt{\RaggedRight} is a command of \texttt{ragged2e}).

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

### 11.4 Use of \{NiceTabular\} with \texttt{threeparttable}

If you wish to use the environment \{NiceTabular\} or \{NiceTabular*\} in an environment \texttt{\texttt{threeparttable}} of the eponymous package, you have to patch the environment \texttt{\texttt{threeparttable}} with the following code:
\AtBeginEnvironment{threeparttable}
{\TPT@hookin{NiceTabular}\TPT@hookin{NiceTabular*}}
\makeatother

The command \AtBeginEnvironment is a command of the package etoolbox which must have been loaded previously.

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{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}

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

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-for-first-row = \scriptstyle \rotate \text{image of } ,}
{code-for-last-col = \scriptstyle }
$A = \begin{pNiceMatrix}[first-row,last-col=4]$
\begin{align*}
e_1 & & e_2 & & e_3 \\
1 & & 2 & & 3 & & e_1 \\
4 & & 5 & & 6 & & e_2 \\
7 & & 8 & & 9 & & e_3
\end{align*}
\end{pNiceMatrix}$

If the command \rotate is used in the “last row” (exterior to the matrix), the corresponding elements are aligned upwards as shown below.
\NiceMatrixOptions{
  code-for-last-row = \scriptstyle \rotate ,
  code-for-last-col = \scriptstyle }
$A = \begin{pNiceMatrix}[last-row=4,last-col=4]
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\text{image of } e_1 & e_2 & e_3
\end{pNiceMatrix}$

12.4 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 \\
\end{bNiceArray}

$\begin{NiceMatrix}
\begin{array}{cccc|c}
1 & -2 & 3 & 4 & 5 \\
0 & 3 & 2 & 1 & 2 & L_2 \gets 2 L_1 - L_2 \\
0 & 1 & 1 & 2 & 3 & L_3 \gets L_1 + L_3
\end{array}
\end{NiceMatrix}$

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 \halign of TeX.

In fact, the option small corresponds to the following tuning:

- the cells of the array are composed with \scriptstyle;
- \arraystretch is set to 0.47;
- \arraycolsep is set to 1.45 pt;
- the characteristics of the dotted lines are also modified.

12.5 The counters iRow and jCol

In the cells of the array, it’s possible to use the LaTeX counters iRow and jCol which represent the number of the current row and the number of the current column\textsuperscript{18}. Of course, the user must not change the value of these counters which are used internally by nicematrix.

In the code-before (cf. p. 9) 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).

\textsuperscript{18}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.
\begin{pNiceMatrix}% don't forget the %
[first-row, 
first-col, 
code-for-first-row = \mathbf{\text{\texttt{\textbackslash alph\{jCol\}}}}, 
code-for-first-col = \mathbf{\text{\texttt{\textbackslash arabic\{iRow\}}}} ] 
\begin{array}{cccc}
 a & b & c & d \\
 1 & 2 & 3 & 4 \\
 2 & 5 & 6 & 7 \\
 3 & 9 & 10 & 11 \\
\end{array}
\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{\textbackslash AutoniceMatrix}, \texttt{\textbackslash pAutoniceMatrix}, \texttt{\textbackslash bAutoniceMatrix}, \texttt{\textbackslash vAutoniceMatrix}, \texttt{\textbackslash vAtoniceMatrix} and \texttt{\textbackslash BAtoniceMatrix}.

These commands take in two mandatory arguments. The first is the format of the matrix, with the syntax \texttt{n-p} where \texttt{n} is the number of rows and \texttt{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).

$C = \pAutoniceMatrix\{3-3\}{C_{\arabic{iRow},\arabic{jCol}}}$

12.6 The option \texttt{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 & b \\
\text{\texttt{\textbackslash \texttt{a}}} & & \\
\text{\texttt{\textbackslash cos a}} & \text{\texttt{\textbackslash \texttt{cos b}}} & \text{\texttt{\textbackslash \texttt{b}}} \\
\end{bNiceMatrix}

It’s possible to change the character used to mark the end of rows with the option \texttt{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 \texttt{verb}}) in the cells of the array.\textsuperscript{19}

\textsuperscript{19}The reason is that, when the option \texttt{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 \texttt{nm-}n-i-j. The command \texttt{\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 \texttt{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.

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

Don’t forget the options \texttt{remember picture} and \texttt{overlay}.

In the \texttt{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.

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

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 nicematrix can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option create-medium-nodes and the second ones with the option create-large-nodes.\(^\text{20}\)

These nodes are not used by 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.

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

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.\(^\text{21}\)

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

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 left-margin and 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 left-margin and right-margin.\(^\text{22}\)

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

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

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

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:

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

\(^{20}\)There is also an option create-extra-nodes which is an alias for the conjunction of create-medium-nodes and create-large-nodes.

\(^{21}\)There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 14).

\(^{22}\)The options left-margin and right-margin take dimensions as values but, if no value is given, the default value is used, which is \text{arraycolsep} (by default: 5 pt). There is also an option margin to fix both left-margin and right-margin to the same value.
Here, we have colored all the cells of the array with `\chessboardcolors`.

Here are the “large nodes” of this array (without use of `margin` nor `extra-margin`).

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

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

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

```latex
\begin{NiceMatrix}
\begin{code-before}=
{ \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{code-before}
1 \ \ \ \ \ \ \ \ \ 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
\end{NiceMatrix}
```
14 API for the developers

The package nicematrix provides two variables which are internal but public:\footnote{According to the LaTeX3 conventions, each variable with name beginning with \texttt{\textbackslash g}\_nicematrix or \texttt{\textbackslash l}\_nicematrix is public and each variable with name beginning with \texttt{\textbackslash g\_\_nicematrix} or \texttt{\textbackslash l\_\_nicematrix} is private.}

- **New 5.2** \texttt{\textbackslash g\_nicematrix\_code\_before\_tl};
- \texttt{\textbackslash g\_nicematrix\_code\_after\_tl}.

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 affectation must be global, allowing to exit the cell, which is a TeX group).

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

**Example** : We want to write a command \texttt{\textbackslash hatchcell} to hatch the current cell (with an optional argument between brackets for the color). It’s possible to program such command \texttt{\textbackslash hatchcell} as follows, explicitly using the public variable \texttt{\textbackslash 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\_west\_lines , pattern\_color = #3 ] ( row - #1 -| col - #2) rectangle ( row - \int_eval:n { #1 + 1 } -| col - \int_eval:n { #2 + 1 } ) ; \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{NiceTabular}{ccc}[hvlines]
Tokyo & Paris & London \\
Roma & \hatchcell[blue!30]Oslo & Miami \\
Los Angeles & Madrid & Roma
\end{NiceTabular}

\begin{tabular}{|c|c|c|}
\hline
Tokyo & Paris & London \\
\hline
Lima & Oslo & Miami \\
\hline
Los Angeles & Madrid & Roma \\
\hline
\end{tabular}
15 Technical remarks

15.1 Definition of new column types

The package \texttt{nicematrix} provides the command \texttt{\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 $\mathbin{?}$ in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job:\textsuperscript{24}:

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

The heavy vertical rule won’t extend in the exterior rows.\textsuperscript{25}

\begin{pNiceArray}{cc?cc}{first-row,last-row=3}
\C_1 & C_2 & C_3 & C_4 \\
a & b & c & d \\e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{pNiceArray}

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

15.2 Diagonal lines

By default, all the diagonal lines\textsuperscript{26} 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 \texttt{\Ddots} in the array can have a marked effect on the final result.

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

Example with parallelization (default):

\begin{verbatim}
A = \begin{pNiceMatrix}
1 & \Cdots & k 1 \\a+b & \textcolor{red}{\Ddots} & \textcolor{red}{k} \textcolor{red}{\texttt{\Vdots}} \\
\textcolor{red}{\Vdots} & \textcolor{red}{\Ddots} & \textcolor{red}{\Vdots} \textcolor{red}{k} \\
a+b & \Cdots \textcolor{red}{a+b} & k 1
\end{pNiceMatrix}
\end{verbatim}

The same example without parallelization:

\begin{verbatim}
A = \begin{pNiceMatrix}
1 & \Cdots & k 1 \\a+b & \textcolor{red}{\texttt{\Vdots}} \textcolor{red}{a+b} & \textcolor{red}{k} 1 \\
a+b & \textcolor{red}{\texttt{\Vdots}} \textcolor{red}{a+b} \textcolor{red}{\texttt{\Vdots}} \textcolor{red}{k} 1
\end{pNiceMatrix}
\end{verbatim}

\textsuperscript{24}The command \texttt{\vrule} is a \TeX{} (and not \LaTeX) command.
\textsuperscript{25}Of course, such rule is defined by the classical technics of \emph{nicematrix} and, for this reason, won’t cross the double rules of \texttt{\hline}.
\textsuperscript{26}We speak of the lines created by \texttt{\Ddots} and not the lines created by a command \texttt{\line} in \texttt{code-after}.
New 5.3  It’s possible to specify the instruction \texttt{\textbackslash Ddots} which will be drawn first (and which will be used to draw the other diagonal dotted line when the parallelization is in force) with the key \texttt{draw-first: \textbackslash Ddots[draw-first]}.

15.3 The “empty” cells

An instruction like \texttt{\textbackslash Ldots}, \texttt{\textbackslash 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 \texttt{\hspace*{1cm}} may be considered as empty.

For \texttt{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 \texttt{\Hspace} (or \texttt{\Hspace*}) is empty. This command \texttt{\Hspace} is a command defined by the package \texttt{nicematrix} with the same meaning as \texttt{\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 \texttt{nicematrix}.

15.4 The option exterior-arraycolsep

The environment \texttt{\{array\}} inserts an horizontal space equal to \texttt{\textbackslash arraycolsep} before and after each column. In particular, there is a space equal to \texttt{\textbackslash arraycolsep} before and after the array. This feature of the environment \texttt{\{array\}} was probably not a good idea\textsuperscript{27}. The environment \texttt{\{matrix\}} of \texttt{amsmath} and its variants (\texttt{\{pmatrix\}}, \texttt{\{vmatrix\}}, etc.) of \texttt{amsmath} prefer to delete these spaces with explicit instructions \texttt{\hskip\textbackslash arraycolsep}\textsuperscript{28}. The package \texttt{nicematrix} does the same in all its environments, \texttt{\{NiceArray\}} included. However, if the user wants the environment \texttt{\{NiceArray\}} behaving by default like the environment \texttt{\{array\}} of \texttt{array} (for example, when adapting an existing document) it’s possible to control this behaviour with the option \texttt{exterior-arraycolsep}, set by the command \texttt{\NiceMatrixOptions}. With this option, exterior spaces of length \texttt{\textbackslash arraycolsep} will be inserted in the environments \texttt{\{NiceArray\}} (the other environments of \texttt{nicematrix} are not affected).

15.5 Incompatibilities

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

\textsuperscript{27}In the documentation of \texttt{amsmath}, we can read: \textit{The extra space of \textbackslash arraycolsep that \texttt{array} adds on each side is a waste so we remove it in \texttt{matrix}} (perhaps we should instead remove it from \texttt{array} in general, but that’s a harder task).

\textsuperscript{28}And not by inserting \texttt{@{}} on both sides of the preamble of the array. As a consequence, the length of the \texttt{\hline} is not modified and may appear too long, in particular when using square brackets.

33
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{\textbackslash stars} similar the well-known commands \texttt{\arabic}, \texttt{\textbackslash alph}, \texttt{\textbackslash Alph}, etc. which produces a number of stars equal to its argument \footnote{In fact: the value of its argument.}

\ExplSyntaxOn
\NewDocumentCommand \stars \o{ m } { \prg_replicate:nn { \value { #1 } } { \star } }
\ExplSyntaxOff

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{\textbackslash widest* = \value{tabularnote}}.

\NiceMatrixOptions
\begin{NiceTabular}{l l r{}}[first-row,code-for-first-row = \bfseries]
\toprule
\text{Last name} & \text{First name} & \text{Birth day} \\
\midrule
\texttt{\textbackslash tabularnote} \texttt{\textbackslash Achard} & \texttt{\textbackslash Jacques} & 5 juin 1962 \\
\texttt{\textbackslash tabularnote} \texttt{\textbackslash Lefebvre} & \texttt{\textbackslash Mathilde} & 23 mai 1988 \\
\texttt{\textbackslash tabularnote} \texttt{\textbackslash Vanesse} & \texttt{\textbackslash Stephany} & 30 octobre 1994 \\
\bottomrule
\end{NiceTabular}
### 16.2 Dotted lines

A permutation matrix (as an example, we have raised the value of `xdots/shorten`).

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

\[
\begin{pmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 \\
1 & 0 & \cdots & \cdots & 0
\end{pmatrix}
\]

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

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

\[
\begin{pmatrix}
1 & \cdots & 1 \\
0 & \cdots & 0 \\
0 & \cdots & \iddots & \iddots & \iddots & \iddots & 0 \\
1 & 0 & \cdots & \cdots & 0
\end{pmatrix}
\]

An example with `\multicolumn`:

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

\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\cdots & 10 \text{ other rows} & \cdots & \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{pmatrix}
\]
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}

An example for the resultant of two polynomials:

\setlength{\extrarowheight}{1mm}
\begin{vNiceArray}{cccc:ccc}[columns-width=6mm]
\addlinespace
a_0 & & & b_0 & & \\
a_1 & \Ddots & & b_1 & \Ddots & \\
\Vdots & \Ddots & & \Vdots & \Ddots & b_0 \\
a_p & \Cdots & a_0 & \Cdots & b_1 & \Cdots & \\
& \Ddots & \a_1 & \b_q & \Cdots & \Vdots \\
& \Vdots & \Ddots & \Vdots & \Cdots & \\
\end{vNiceArray}

An example for a linear system:

$\begin{pNiceArray}{*6c|c}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
1 & 1 & 1 & \Cdots & & 1 & 0 \\
0 & 1 & 0 & \Cdots & & 0 & L_2 \gets L_2-L_1 \\
0 & 0 & 1 & \Ddots & & \Vdots & L_3 \gets L_3-L_1 \\
& \ & \ & \ & \Vdots & \ & \Vdots \\
\Vdots & & \ & \ & 0 & \ & \ & \Vdots & \ & \Vdots \\
0 & \ & \ & \ & \ & \ & \ & \ & \ & \ & \ & L_n \gets L_n-L_1
\end{pNiceArray}$
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}

$\begin{pNiceMatrix}[first-row,first-col]
& & \ldots & 1 \text{ columns} \\
& 1 & 1 & 1 & \ldots & 1 \\
& 0 & 1 & 0 & \ldots & 0 \\
& 0 & 0 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots \\
& 0 & \ldots & 0 & 1 & 0 \\
\end{pNiceMatrix}$

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]
\begin{NiceMatrixOptions}
  \{last-col,code-for-last-col = \color{blue}\scriptstyle,light-syntax\}
\end{NiceMatrixOptions}
\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 { L_2 \gets -2 L_1 + L_2 } \\
0 & 6 & 24 & 78 & 33 { L_3 \gets -3 L_1 + L_3 } \\
0 & 12 & 60 & 252 & 96 { L_4 \gets -4 L_1 + L_4 }
\end{pNiceArray}$

...
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:

```latex
\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 that the nodes created by nicematrix).

```latex
\begin{pNiceArray}{>{\strut}cccc}
\create-large-nodes,margin,extra-margin = 2pt
a_{11} & a_{12} & a_{13} & a_{14} \\
\hline
a_{21} & a_{22} & a_{23} & a_{24} \\
\hline
a_{31} & a_{32} & a_{33} & a_{34} \\
\hline
a_{41} & a_{42} & a_{43} & a_{44} \\
\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 \hline, see the remark p. 6.}
It’s possible to color a row with \rowcolor in the code-before (or with \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 fit. This Tikz node is filled after the construction of the matrix. In order to see the text under this node, we have to use transparency with the blend mode equal to multiply.

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

\begin{bNiceMatrix}
\[
\begin{array}{cccc}
0 & \Cdots & 0 \\
1 & \Cdots & 1 \\
0 & \Cdots & 0
\end{array}
\end{bNiceMatrix}

This code fails with latex-dvips-ps2pdf because Tikz for dvips, as for now, doesn’t support blend modes. However, the following code, in the preamble, should activate blend modes in this way of compilation.
\ExplSyntaxOn
\makeatletter
\tl_set:Nn \l_tmpa_tl {pgfsys-dvips.def}
\tl_if_eq:NNT \l_tmpa_tl \pgfsysdriver
  {\cs_set:Npn\pgfsys@blend@mode#1{\special{ps:~/\tl_upper_case:n #1~.setblendmode}}}
\makeatother
\ExplSyntaxOff

We recall that, for a rectangle of merged cells (with the command \Block), a Tikz node is created for the set of merged cells with the name i-j-block where $i$ and $j$ are the number of the row and the number of the column of the upper left cell (where the command \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 -medium.

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

Consider now the following matrix which we have named example.

\begin{pNiceArray}{ccc}
\[
\begin{array}{ccc}
a & a + b & a + b + c & L_1 \\
 a & a & a + b & L_2 \\
 a & a & a & L_3
\end{array}
\end{pNiceArray}
\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

We obtain the following matrix.

\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

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

\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

We obtain the following matrix.

\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

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

\begin{pNiceArray}{>{\strut}cccc}
| 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
\tikz \path [name suffix = -large,fill = red!15, blend mode = multiply] 
(1-1.north west) 
|-- (2-2.north west) 
|-- (3-3.north west) 
|-- (4-4.north west) 
|-- (4-4.south east) 
|-- (1-1.north west) ;
\end{pNiceArray}
16.6 Direct use of the Tikz nodes

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

The use of \code{\NiceMatrixBlock} with the option \code{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}
\begin{array}{cc}
& \\
\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} \\
&
\end{array}
\end{NiceMatrixBlock}

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

\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}
\end{array}$
\end{NiceMatrixBlock}

\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`, `\dots`, `\vdots`, and `\ddots` are redefined in the environments provided by `nicematrix` as explained previously. In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `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 `\ialign` to begin the `\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](http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf)

First, we load `pgfcore` and the module `shapes`.

```latex
\RequirePackage{pgfcore}
\usepgfmodule{shapes}
```

We give the traditional declaration of a package written with `expl3`:

```latex
\RequirePackage{13keys2e}
\ProvidesExplPackage{nicematrix}
\myfiledate
\myfileversion
\Enhanced arrays with the help of PGF/TikZ
```

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

We load some packages.

```latex
\RequirePackage{array}
\RequirePackage{amsmath}
\RequirePackage{sparse}
```

42
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 \tikzpicture-\endtikzpicture (or \begin{tikzpicture}-\end{tikzpicture}) 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 sparse, and so, if the command \iddots has already been defined (for example by the package mathdots), we don't define it again.

\ProvideDocumentCommand \iddots { }
This definition is a variant of the standard definition of \textdots.

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).

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

We have to know whether colortbl is loaded in particular for the redefinition of \everycr.

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

The command \texttt{\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 \texttt{\CT@drs}.

Idem for \texttt{\CT@arc}.

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

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

Idem for \texttt{\CT@arc}.
\cs_set:Npn \hline
{\ifnum0=\fi\cs_set_eq:NN \hskip \vskip\cs_set_eq:NN \vrule \hrule\cs_set_eq:NN \@width \@height\CT@arc@\leaders\hrule\@height\hfill\skip_horizontal:N \c_zero_dim\leaders \hrule \@height \arrayrulewidth \hfill\skip_vertical:N -\arrayrulewidth}

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 {\@@_cline_i:en\l_@@_first_col_int}

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 } { \multispan { \int_eval:n { #2 - #1 } } \hfill }\multispan { \int_eval:n { #3 - #2 + 1 } }\CT@arc@\leaders \hrule \@height \arrayrulewidth \hfill\skip_horizontal:N \c_zero_dim\leaders \hrule \@height \arrayrulewidth \hfill\skip_vertical:N -\arrayrulewidth}

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

See question 99041 on TeX StackExchange.

45
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_generate_variant:Nn \@@_cline_i:nn { e n }
\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 }
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_set_CT@arc@: { \peek_meaning:NTF \[ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
\end{verbatim}

\begin{verbatim}
\ CS_new:Npn \@@_tab_or_array_colsep: { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
\end{verbatim}

The column \texttt{S} of \texttt{siunitx}

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

\begin{verbatim}
\bool_new:N \c_@@_siunitx_loaded_bool
\AtBeginDocument
  \{ \bool_if:NT \l_@@_siunitx_loaded_bool \c_@@_siunitx_loaded_bool \}
\end{verbatim}

The command \texttt{\NC@rewrite@S} is a \LaTeX{} command created by \texttt{siunitx} in connection with the \texttt{S} column. In the code of \texttt{siunitx}, this command is defined by:

\begin{verbatim}
\renewcommand*{\NC@rewrite@S}{[1][]}
\end{verbatim}

We want to patch this command (in the environments of \texttt{nicematrix}) in order to have:

\begin{verbatim}
\renewcommand*{\NC@rewrite@S}{[1][]}
\end{verbatim}

46
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 (\texttt{\group_begin:\texttt{-}\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 \texttt{\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}
166 \cs_set_protected:Npn \@@_adapt_S_column:
167 { 168 \bool_if:NT \c_@@_siunitx_loaded_bool
169  { 170 \group_begin:
171 \@temptokena = { }
172 \cs_set_eq:NN \NC@find \prg_do_nothing:
173 \NC@rewrite@S { }
174 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 \texttt{V} for \texttt{\tl_gset:NV}).
175 \tl_gset:NV \g_tmpa_tl \@temptokena
176 \group_end:
177 \tl_new:N \c_@@_table_collect_begin_tl
178 \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
179 \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
180 \tl_new:N \c_@@_table_print_tl
181 \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
182 The token lists \c_@@_table_collect_begin_tl and \c_@@_table_print_tl contain now the two commands of \siunitx.
183 If the adaptation has been done, the command \@@_adapt_S_column: becomes no-op (globally).
184 \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
185 }
186 }
187 \AtBeginDocument
188 { 189 \bool_if:nTF { ! \c_@@_siunitx_loaded_bool } 190 { \cs_set_eq:NN \@@_adapt_S_column: \prg_do_nothing: }
191 { 192 \cs_new_protected:Npn \@@_adapt_S_column: 193 { 194 \cs_set_protected:Npn \@@_adapt_S_column: 195 { 196 \renewcommand*{\NC@rewrite@S}[^{1}][{ ] 197 { \@temptokena \exp_after:wN
198 { \tex_the:D \@temptokena
199 > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
\@_true_c: will be replaced statically by c at the end of the construction of the preamble.
200 \@_true_c: 201 < { \c_@@_table_print_tl \@@_end_Cell: } 202 }
\NC@find
203 }
204 }
\end{verbatim}

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

\begin{verbatim}
\regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
  \begin{lrbox}{\@mainaux}
  \ExplSyntaxOn
  \cs_if_free:NT \pgfsyspdfmark { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
  \ExplSyntaxOff
  \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
\end{lrbox}
\end{verbatim}

\section*{Parameters}

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

\begin{verbatim}
\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
\end{verbatim}

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

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

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

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

The command \texttt{\textbackslash NiceMatrixLastEnv} is not used by the package \texttt{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 \texttt{must} be expandable since it will be used in \texttt{pgf} nodes.

\begin{verbatim}
\NewExpandableDocumentCommand \NiceMatrixLastEnv { } { \int_use:N \g_@@_env_int }
\end{verbatim}

The following command is only a syntaxic shortcut. The \texttt{q} in \texttt{qpoint} means \textit{quick}.

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

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

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

\begin{verbatim}
\dim_new:N \l_@@_columns_width_dim
\end{verbatim}

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

\begin{verbatim}
\seq_new:N \g_@@_names_seq
\end{verbatim}

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

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

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

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

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

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

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

\begin{verbatim}
\dim_new:N \l_@@_tabular_width_dim
\end{verbatim}

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

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

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

\begin{verbatim}
\colorlet { nicematrix-last-col } { . }
\colorlet { nicematrix-last-row } { . }
\end{verbatim}

The following string is the name of the current environment or the current command of \texttt{nicematrix} (despite its name which contains \texttt{env}).

\begin{verbatim}
\str_new:N \g_@@_name_env_str
\end{verbatim}

The following string will contain the word \texttt{command} or \texttt{environment} whether we are in a command of \texttt{nicematrix} or in an environment of \texttt{nicematrix}. The default value is \texttt{environment}.

\begin{verbatim}
\tl_set:Nn \g_@@_com_or_env_str { environment }
\end{verbatim}

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains \texttt{env}). This command must \textit{not} be protected since it will be used in error messages.

\begin{verbatim}
\cs_new_protected:Npn \@@_full_name_env:
\{ \\
\str_if_eq:VnTF \g_@@_com_or_env_str { command } \\
{ command \space \c_backslash_str \g_@@_name_env_str } \\
{ environment \space \{ \g_@@_name_env_str \} }
\}
\end{verbatim}
The following token list corresponds to the option `code-after` (it’s also possible to set the value of that parameter with the command `\CodeAfter`).

\tl_new:N \g_nicematrix_code_after_tl

The following token list has a function similar to \g_nicematrix_code_after_tl but it is used internally by nicematrix. In fact, we have to distinguish between \g_nicematrix_code_after_tl and \g@@_internal_code_after_tl because we must take care of the order in which instructions stored in that parameters are executed.

\tl_new:N \g@@_internal_code_after_tl

The counters \l@@_old_iRow_int and \l@@_old_jCol_int will be used to save the values of the potential LaTeX counters \iRow and \jCol. These LaTeX counters will be restored at the end of the environment.

\int_new:N \l@@_old_iRow_int
\int_new:N \l@@_old_jCol_int

The TeX counters \c@iRow and \c@jCol will be created in the beginning of \{NiceArrayWithDelims\} (if they don’t exist previously).

The following token list corresponds to the key `rules/color` available in the environments.

\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 creates two other 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

Variables for the exterior rows and columns

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

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

\int_new:N \l_@@_first_row_int
\int_set:Nn \l_@@_first_row_int 1

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

\int_new:N \l_@@_first_col_int
\int_set:Nn \l_@@_first_col_int 1
• Last row

The counter \l_@@_last_row_int is the number of the potential “last row”, as specified by the key last-row. A value of \(-2\) means that there is no “last row”. A value of \(-1\) means that there is a “last row” but we don’t know the number of that row (the key 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 \{pNiceArray\}, the option 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 \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 \(-2\) means that there is no last column. A value of \(-1\) means that we are in an environment without preamble (e.g. \{bNiceMatrix\}) and there is a last column but we don’t know its value because the user has used the option last-col without value. A value of 0 means that the option last-col has been used in an environment with preamble (like \{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}

However, we have also a boolean. Consider the following code:

\begin{verbatim}
\begin{pNiceArray}{cc}{last-col}
1 & 2 \\
3 & 4
\end{pNiceArray}
\end{verbatim}

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.

\begin{verbatim}
\bool_new:N \g_@@_last_col_found_bool
\end{verbatim}

This boolean is set to \texttt{false} at the end of \@@_pre_array.

The command \texttt{tabularnote}

The \LaTeX{} counter \texttt{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 \texttt{refstepcounter} in order to have the tabular notes referenceable.

\begin{verbatim}
\newcounter { tabularnote }
\end{verbatim}

\footnote{We can’t use \l_@@_last_row_int for this usage because, if \texttt{nicematrix} has read its value from the aux file, the value of the counter won’t be \(-1\) any longer.}
We will store in the following sequence the tabular notes of a given array.

\seq_new:N \g_@@_tabularnotes_seq

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 notes are composed as a comma separated list (e.g. \emph{a,b,c}).

\int_new:N \l_@@_number_of_notes_int

The following function can be redefined by using the key \texttt{notes/style}.

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

The following function can be redefined by using the key \texttt{notes/label-in-tabular}.

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

The following function can be redefined by using the key \texttt{notes/label-in-list}.

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

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

\cs_set:Npn \thetabularnote { \textsuperscript { \@@_notes_style:n { \textit { \textsuperscript { \alph { \@@_number_of_notes_int } } } } } }

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

\AtBeginDocument

\bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
{ \NewDocumentCommand \tabularnote { m } { \@@_error:n { enumitem~not~loaded } } }
{ \@@_error:n { enumitem-not-loaded } }

The type of list \texttt{tabularnotes} will be used to format the tabular notes at the end of the array in the general case and \texttt{tabularnotes*} will be used if the key \texttt{para} is in force.

\newlist { tabularnotes } { enumerate } { 1 }
\setlist [ tabularnotes ]
{ noitemsep , leftmargin = * , align = left , labelsep = Opt ,
  label = \textsuperscript { \@@_notes_label_in_list:n { \@@_notes_style:n { \textit { \textsuperscript { \alph { \@@_number_of_notes_int } } } } } } ,
}
\newlist { tabularnotes* } { enumerate* } { 1 }
\setlist [ tabularnotes* ]
{ afterlabel = \nobreak ,
  itemjoin = \quad ,
  label = \textsuperscript { \@@_notes_label_in_list:n { \@@_notes_style:n { \textit { \textsuperscript { \alph { \@@_number_of_notes_int } } } } } } }

53
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 \@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{We should try to find a solution to that problem.}

\NewDocumentCommand \tabularnote { m } {
  \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool } {
    \@@_error:n { tabularnote-forbidden } }
  {
    \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. a, b, c).
    \int_incr:N \l_@@_number_of_notes_int
    We expand the content of the note at the point of use of \tabularnote as does \footnote.
    \seq_gput_right:Nx \g_@@_tabularnotes_seq { #1 }
    \peek_meaning:NF \tabularnote
    {
      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.
      \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.
      \@@_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.
      \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).
\skip_horizontal:n { \box_wd:N \l_tmpa_box }
}
Command for creation of rectangle nodes

The following command should be used in a \texttt{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.

\begin{verbatim}
\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 } } }
\end { pgfscope }
\end{verbatim}

The command \texttt{\@@_pgf_rect_node:nnn} is a variant of \texttt{\@@_pgf_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
\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 } } }
\pgfnode
{ rectangle }
{ center }
{ \vbox_to_ht:nn { \dim_abs:n { #3 - #2 } } { \vfill \hbox_to_wd:nn { \dim_abs:n { #5 - #3 } } { } } }
\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.

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

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

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.

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).

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.

The token list `\_@@_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.

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

The string `\_@@_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).

The flag `\_@@_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`).
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 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 ":".

\t_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 { \cs_if_exist_p:N \tikzpicture } { \str_if_eq_p:nn { #1 } { standard } } { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } } { \@@_error:n { bad-option-for-line-style } } , line-style .value_required:n = true , color .tl_set:N = \l_@@_xdots_color_tl , color .value_required:n = true , shorten .dim_set:N = \l_@@_xdots_shorten_dim , shorten .value_required:n = true , down .tl_set:N = \l_@@_xdots_down_tl , up .tl_set:N = \l_@@_xdots_up_tl , draw-first .code:n = \prg_do_nothing: , unknown .code:n = \@@_error:n { Unknown-option-for-xdots } }

\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.

\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 ,

58
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
}
parallelize-diags .bool_set:N = \l_@@_parallelize_diags_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
}

We define a set of keys used by the environments of nicematrix (but not by the command \NiceMatrixOptions).
\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
} ,

59
The options \texttt{c}, \texttt{t} and \texttt{b} of the environment \texttt{NiceArray} have the same meaning as the option of the classical environment \texttt{array}.

\begin{verbatim}
\tl_set:Nn \l_@@_baseline_str c , \tl_set:Nn \l_@@_baseline_str t , \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 } { \dim_set:Nn \l_@@_columns_width_dim { #1 } } , columns-width .value_required:n = true , name .code:n = \legacy_if:nF { measuring@ } \{ \str_set:Nn \l_tmpa_str { #1 } \} \{ \dim_set:Nn \l_@@_auto_columns_width_dim { #1 } \} , columns-width .value_required:n = true , name .value_required:n = true , code-after .tl_gset:N = \g_nicematrix_code_after_tl , 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
\end{verbatim}

We test whether we are in the measuring phase of an environment of \texttt{amsmath} (always loaded by \texttt{nicematrix}) because we want to avoid a fallacious message of duplicate name in this case.

\begin{verbatim}
\legacy_if:nF { measuring@ } \{ \str_set:Nn \l_tmpa_str { #1 } \} \{ \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str { \@@_error:nn { Duplicate-name } { #1 } } \} \{ \str_set_eq:NN \l_@@_name_str \l_tmpa_str \} , name .value_required:n = true , code-after .tl_gset:N = \g_nicematrix_code_after_tl , 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
\end{verbatim}

\begin{verbatim}
\keys_define:nn { NiceMatrix / notes } \{ para .bool_set:N = \l_@@_notes_para_bool , para .default:n = true , code-before .tl_set:N = \l_@@_notes_code_before_tl , code-before .value_required:n = true , code-after .tl_set:N = \l_@@_notes_code_after_tl , code-after .value_required:n = true , bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool , bottomrule .default:n = true , style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } , style .value_required:n = true , label-in-tabular .code:n = \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } , label-in-tabular .value_required:n = true , label-in-list .code:n = \cs_set:Nn \@@_notes_label_in_list:n { #1 } , label-in-list .value_required:n = true , enumitem-keys .code:n = \bool_if:NTF \c_@@_in_preamble_bool
\end{verbatim}
We begin the construction of the major sets of keys (used by the different user commands and environments).

\keys_define:nn { NiceMatrix }
{
  \NiceMatrixOptions .inherit:n =
  \NiceMatrix / Global ,
  \NiceMatrixOptions / xdots .inherit:n = \NiceMatrix / xdots ,
  \NiceMatrixOptions / rules .inherit:n = \NiceMatrix / rules ,
  \NiceMatrixOptions / notes .inherit:n = \NiceMatrix / notes ,
  \NiceMatrix .inherit:n =
  \NiceMatrix / Global ,
  \NiceMatrix / Env ,
  \NiceMatrix / xdots .inherit:n = \NiceMatrix / xdots ,
  \NiceMatrix / rules .inherit:n = \NiceMatrix / rules ,
  \NiceTabular .inherit:n =
  \NiceTabular / Global ,
  \NiceTabular / Env ,
  \NiceTabular / xdots .inherit:n = \NiceMatrix / xdots ,
  \NiceTabular / rules .inherit:n = \NiceMatrix / rules ,
  \NiceArray .inherit:n =
  \NiceMatrix / Global ,
  \NiceMatrix / Env ,
  \NiceMatrix / xdots .inherit:n = \NiceMatrix / xdots ,
  \NiceMatrix / rules .inherit:n = \NiceMatrix / rules ,
  \pNiceArray .inherit:n =
We finalise the definition of the set of keys “\texttt{NiceMatrix / NiceMatrixOptions}” with the options specific to \texttt{NiceMatrixOptions}.

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

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

\begin{verbatim}
renew-matrix .code:n = \@@_renew_matrix: ,
renew-matrix .value_forbidden:n = true ,
\end{verbatim}

The key \texttt{transparent} is now considered as obsolete (because its name is ambiguous).

\begin{verbatim}
transparent .meta:n = { renew-dots , renew-matrix } ,
transparent .value_forbidden:n = true ,
\end{verbatim}

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

\begin{verbatim}
exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
\end{verbatim}

If the option \texttt{columns-width} is used, all the columns will have the same width.

\begin{verbatim}
columns-width .code:n =
  \tl_if_eq:nnTF { #1 } { auto }
    { \@@_error:n { Option~auto~for~columns-width } }
    { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
\end{verbatim}

Usually, an error is raised when the user tries to give the same name to two distincts environments of nicematrix (theses names are global and not local to the current \TeX{} scope). However, the option \texttt{allow-duplicate-names} disables this feature.

\begin{verbatim}
allow-duplicate-names .code:n =
  \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
allow-duplicate-names .value_forbidden:n = true ,
\end{verbatim}

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

\begin{verbatim}
letter-for-dotted-lines .code:n =
  \tl_if_single_token:nTF { #1 }
    { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } ,
    { \@@_error:n { Bad~value~for~letter~for~dotted~lines } }
  \\dim_set:Nn \l_@@_columns_width_dim { #1 } ,
\end{verbatim}
\texttt{\textbackslash NiceMatrixOptions} is the command of the \texttt{nicematrix} package to fix options at the document level. The scope of these specifications is the current TeX group.

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

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

In the environments \texttt{\{NiceArray\}} and its variants, the option \texttt{last-col} must be used without value because the number of columns of the array is read from the preamble of the array.

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

63
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: {
  \int_gincr:N \c@jCol
  \int_compare:nNnTF \c@iRow = 0 {
    \int_compare:nNnT \c@jCol > 0 {
      \l_@@_code_for_first_row_tl 
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

The content of the cell is composed in the box \l_@@_cell_box because we want to compute some dimensions of the box. The \bbox_set:Nw will be in the \@@_end_Cell: (and the potential \c_math_toggle_token also).

\bbox_set:Nw \l_@@_cell_box
\bool_if:NF \l_@@_NiceTabular_bool {
  \c_math_toggle_token
  \bool_if:NT \l_@@_small_bool \scriptstyle
}

We will call corners of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn’t always exist simultaneously).

The codes \l_@@_code_for_first_row_tl and \l_@@_code_for_last_row_tl don’t apply in the corners of the matrix.

\int_compare:nNnTF \c@iRow = 0 {
  \int_compare:nNnT \c@jCol > 0 {
    \l_@@_code_for_first_row_tl
    \xglobal \colorlet { nicematrix-first-row } { . }
  }
}

\int_compare:nNnT \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.
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 \texttt{last-row}, some lines of code will be dynamically added to this command.

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”).

The following computations are for the “first row” and the “last row”.

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 \_@@_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 \rlap, a \llap or a \mathclap of mathtools.

- the cells with a command \Ldots or \Cdots, \Vdots, etc., should also be considered as empty: if 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 code-after); however, if nullify-dots is not in force, a phantom of \ldots, \cdots, \vdots is inserted and its width is not equal to zero; that’s why these commands raise a boolean \_@@_empty_cell_bool and we begin by testing this boolean.

```latex
\bool_if:NTF \_@@_empty_cell_bool
{ \box_use_drop:N \_@@_cell_box }
\dim_compare:nNnTF { \box_wd:N \_@@_cell_box } > \c_zero_dim
{ \_@@_node_for_the_cell: }
\bool_gset_false:N \_@@_empty_cell_bool
```

The following command creates the PGF name of the node with, of course, \_@@_cell_box as the content.

```latex
\cs_new_protected:Npn \_@@_node_for_the_cell: 
{ \pgfpicture
  \pgfsetbaseline \c_zero_dim
  \pgfrememberpicturepositiononpagetrue
  \pgfset {
    inner~sep = \c_zero_dim ,
    minimum~width = \c_zero_dim
  }
  \pgfnode { rectangle } { base }
  \{ \box_use_drop:N \_@@_cell_box \}
  \{ \_@@_env: - \int_use:N \_@@_env \_@@_env \}
  \str_if_empty:NF \_@@_name_str
  { \pgfnodealias
    \{ \_@@_name_str - \int_use:N \_@@_env \_@@_env \}
    \{ \_@@_name_str - \int_use:N \_@@_env \_@@_env \}
  }
  \endpgfpicture
```

The second argument of the following command \_@@_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 \_@@_type_lines_tl the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```latex
\begin{pNiceMatrix}
  1 & 2 & 3 & 4 \\ \\
  5 & \Cdots & & 6 \\
  7 & \Cdots[color=red] & \\
\end{pNiceMatrix}
```

the content of \_@@_Cdots_lines_tl will be:

\begin{verbatim}
\begin{Verbatim}
\end{Verbatim}
\end{verbatim}
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.

\cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
{
It's important to use a \tl_gput_right:cx and not a \tl_gput_left:cx because we want the \Ddots lines to be drawn in the order of appearance in the array (for parallelisation).

\bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
{ g_@@_ #2 _ lines _ tl }
{ \use:c { \@@_draw_ #2 : nnn }
  \int_use:N \c@iRow
  \int_use:N \c@jCol
  \exp_not:n { #3 } }
}

We want to use \array of \array. However, if the class used is revtex4-1 or revtex4-2, we have to do some tuning and use the command \@array@array instead of \array because these classes do a redefinition of \array incompatible with our use of \array.

\cs_new_protected:Npn \@@_revtex_array:
{ \cs_set_eq:NN \@acoll \@arrayacol
  \cs_set_eq:NN \@acolr \@arrayacol
  \cs_set_eq:NN \@acol \@arrayacol
  \cs_set_nopar:Npn \@halignto { }
  \@array@array }
\cs_new_protected:Npn \@@_array:
{ \bool_if:NTF \c_@@_revtex_bool \@@_revtex_array:
  { \bool_if:NTF \l_@@_NiceTabular_bool
    { \dim_set_eq:NN \col@sep \tabcolsep }
    { \dim_set_eq:NN \col@sep \arraycolsep }
    \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
    { \cs_set_nopar:Npn \@halignto { } }
    { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
  }
  \@tabarray }
\{ \str_if_eq:VnTF \l_@@_baseline_str c c t \}
\l_@@_baseline_str may have the value t, c or b. However, if the value is b, we compose the \array (of \array) with the option t and the right translation will be done further.

\cs_set_eq:NN \@@_old_ialign: \ialign
The following command creates a row node (and not a row of nodes!).

\cs_new_protected:Npn \@@_create_row_node:
The \hbox (or \hbox:n) is mandatory.
\begin{verbatim}
\hbox
{\
  \bool_if:NT \l_@@_code_before_bool
  {\
    \vtop
    {\
      \skip_vertical:N 0.5\arrayrulewidth\
      \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
      \skip_vertical:N -0.5\arrayrulewidth
    }
  }
  \pgfpicture
  \pgfrememberpicturepositiononpagetrue
  \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
  \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth }
\str_if_empty:NF \l_@@_name_str
  {\
    \pgfnodealias
    { \l_@@_name_str - row - \int_use:N \c@iRow }
  }
\endpgfpicture
}
\end{verbatim}

The following must not be protected because it begins with \noalign.
\begin{verbatim}
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i: 
{\int_gzero:N \c@jCol
  \bool_if:NF \g_@@_row_of_col_done_bool
  {\
    \@@_create_row_node:
  }
\bool_if:NT \l_@@_hlines_bool
  {\
    \CT@arc@ is a command of\colortbl which sets the color of the rules in the array. The\nicematrix uses it even if \colortbl is not loaded. We use a TeX group in order to limit the scope of \CT@arc@.
    {\hrule height \arrayrulewidth width \c_zero_dim }
  }
}
\end{verbatim}

We don’t draw the rules of the key hlines (or hvlines) but we reserve the vertical space for theses rules.
\begin{verbatim}
\bool_if:NT \l_@@_hlines_bool
{\
  \int_compare:nNnT \c@iRow > { -1 }
  {\
    \int_compare:nNnF \c@iRow = \l_@@_last_row_int
    {\
      \\@@_create_row_node:
    }
  }
}
\end{verbatim}

The counter \c@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}
\CS@newcolumntype is the command \newcolumntype of array without the warnings for redefinitions of columns types (we will use it to redefine the columns types \(\mathbb{W}\) and \(\mathbb{W}^{-}\)).
\cs_set_protected:Npn \@@_newcolumntype #1 
{\
  \cs_set:cpn { NC \find @ #1 } ##1 #1 { \NC@ { ##1 } }
\peek_meaning:NTF {\[}
  {\newcol@ #1 }
}
\end{verbatim}

The command \@@_newcolumntype is the command \newcolumntype of array without the warnings for redefinitions of columns types (we will use it to redefine the columns types \(\mathbb{W}\) and \(\mathbb{W}^{-}\)).
\begin{verbatim}
\cs_set_protected:Npn \@@_newcolumntype #1 
{\
  \cs_set:cpn { NC \find @ #1 } ##1 #1 { \NC@ { ##1 } }
\peek_meaning:NTF {\[}
  {\newcol@ #1 }
}
\end{verbatim}
When the key `renew-dots` is used, the following code will be executed.

```latex
\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:}
```

When the key `colortbl-like` is used, the following code will be executed.

```latex
\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}
```

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

```latex
\cs_new_protected:Npn \@@_pre_array:
{\tl_put_left:Nn \@BTnormal \@@_create_row_node: }
\box_clear_new:N \l_@@_cell_box
\cs_if_exist:NT \theiRow
{ \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
\int_gzero_new:N \c@iRow
\cs_if_exist:NT \thejCol
{ \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
\int_gzero_new:N \c@jCol
\normalbaselines
```

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

```latex
\bool_if:NT \l_@@_small_bool
{\cs_set_nopar:Npn \arraystretch { 0.47}
\dim_set:Nn \arraycolsep { 1.45 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.
The box \texttt{@arstrutbox} is a box constructed in the beginning of the environment \texttt{array}. The construction of that box takes into account the current values of \texttt{arraystretch} and \texttt{extrarowheight} (of \texttt{array}). That box is inserted (via \texttt{@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 \texttt{@arstrutbox} and that's why we do it in the \texttt{\ialign}.

\begin{verbatim}
\dim_gzero_new:N \g_@@_dp_row_zero_dim
\dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
\dim_gzero_new:N \g_@@_ht_row_zero_dim
\dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
\dim_gzero_new:N \g_@@_ht_row_one_dim
\dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
\dim_gzero_new:N \g_@@_dp_row_zero_dim
\dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
\dim_gzero_new:N \g_@@_dp_last_row_dim
\dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
\dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
\end{verbatim}

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

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

\begin{verbatim}
\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
\cs_set_eq:NN \@@_old_iddots \iddots
\bool_if:NTF \l_@@_standard_cline_bool
{ \cs_set_eq:NN \cline \@@_standard_cline }
{ \cs_set_eq:NN \cline \@@_cline }
\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 \hdottedline \@@_hdottedline:
\cs_set_eq:NN \Hline \@@_Hline:
\cs_set_eq:NN \Hspace \@@_Hspace:
\cs_set_eq:NN \Hdottedline \@@_Hdottedline:
\cs_set_eq:NN \Block \@@_Block:
\cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
\end{verbatim}

\footnote{The option \texttt{small} of \texttt{nicematrix} changes (among other) the value of \texttt{arraystretch}. This is done, of course, before the call of \texttt{\{array\}).}
The sequence \texttt{g@@multicolumn\_cells\_seq} will contain the list of the cells of the array where a command 	exttt{\multicolumn{n}{...}{...}} with \( n > 1 \) is issued. In \texttt{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 \texttt{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 \texttt{array}, \texttt{c@iRow} will be the total number de rows. \texttt{g@@row\_total\_int} will be the number or rows excepted the last row (if \texttt{\l@@last\_row\_bool} has been raised with the option \texttt{last-row}).

The counter \texttt{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 \texttt{g@@col\_total\_int}. These counters are updated in the command \texttt{@@\_Cell}: executed at the beginning of each cell.

During the construction of the array, the instructions \texttt{\Cdots}, \texttt{\Ldots}, etc. will be written in token lists \texttt{g@@Cdots\_lines\_tl}, etc. which will be executed after the construction of the array.

This is the end of \texttt{@@\_pre\_array}.

The environment \texttt{\{NiceArrayWithDelims\}}

\begin{verbatim}
\NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } } {
 \@@\provide\pgf\sys\pdf\mark:
 \bool_if:NT \c@@footnote_bool \savenotes
 The aim of the following \texttt{\begroup} (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.

 \begroup
 \tl_set:Nn \l@@left\_delim\_tl { #1 }
 \tl_set:Nn \l@@right\_delim\_tl { #2 }
 \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:
\end{verbatim}
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 beginning of arrays done by colorbl. Of course, we restore the value of \CT@arc@ at the end of our environment.

We deactivate Tikz externalization because we will use PGF pictures with the options overlay and remember picture (or equivalent forms).

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

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

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

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 t, c, b and baseline.

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).

\footnotesize{\color{rgb}\{0.5,0.5,0\}}
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.

\int_compare:nNnF \l_@@_last_row_int = \{ -2 \}
\int_decr:N \c@iRow
\int_compare:nNnF \l_@@_last_col_int = \{ -2 \}
\int_decr:N \c@jCol

Now, we will create all the \texttt{col} nodes and \texttt{row} nodes with the informations written in the \texttt{aux} file.

You use the technique described in the page 1229 of \texttt{pgfmanual.pdf}, version 3.1.4b.

\pgfsys@markposition { \@@_env: - position }
\pgfsys@getposition { \@@_env: - position } \@@_picture_position:
\pgfpicture

First, the creation of the \texttt{row} nodes.

\int_step_inline:nnn
{ \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int_seq } 1 }
{ \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int_seq } 2 + 1 }
\pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
\pgfcoordinate { \@@_env: - row - ##1 }
{ \pgfpointdiff \@@_picture_position: \@@_node_position: }

Now, the creation of the \texttt{col} nodes.

\int_step_inline:nnn
{ \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int_seq } 3 }
{ \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int_seq } 4 + 1 }
\pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
\pgfcoordinate { \@@_env: - col - ##1 }
{ \pgfpointdiff \@@_picture_position: \@@_node_position: }
\endpgfpicture
\group_begin:
\bool_if:NT \c_@@_tikz_loaded_bool
{\tikzset
{every_picture / .style =
{ overlay , name-prefix = \@@_env: - }
}
\cs_set_eq:NN \cellcolor \@@_cellcolor
\cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
\cs_set_eq:NN \rowcolor \@@_rowcolor
\cs_set_eq:NN \rowcolors \@@_rowcolors
\cs_set_eq:NN \columncolor \@@_columncolor
\cs_set_eq:NN \chessboardcolors \@@_chessboardcolors

We compose the \texttt{code-before} in math mode in order to nullify the spaces put by the user between instructions in the \texttt{code-before}.

\bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
\l_@@_code_before_tl
\bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
\group_end:
}
A value of $-1$ for the counter $\texttt{l@@_last_row_int}$ means that the user has used the option \texttt{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).

\begin{verbatim}
\int_compare:nNnT \l@@_last_row_int > {-2}
{\tl_put_right:Nn \@@_update_for_first_and_last_row:
  \dim_gset:Nn \g_@@_ht_last_row_dim
  \dim_max:nn \g_@@_ht_last_row_dim \{ \box_ht:N \l@@_cell_box \}
  \dim_gset:Nn \g_@@_dp_last_row_dim
  \dim_max:nn \g_@@_dp_last_row_dim \{ \box_dp:N \l@@_cell_box \}
}
\int_compare:nNnT \l@@_last_row_int = {-1}
{\bool_set_true:N \l@@_last_row_without_value_bool
  \str_if_empty:NTF \l@@_name_str
  \cs_if_exist:cT \{@@_last_row_\int_use:N \g_@@_env_int
  \int_set:Nn \l@@_last_row_int
  \use:c \{@@_last_row_\int_use:N \g_@@_env_int \}
  \}
}
\str_if_empty:NTF \l@@_name_str
{\cs_if_exist:cT \{@@_last_row_\l_00_name_str
  \int_set:Nn \l@@_last_row_int
  \use:c \{@@_last_row_\l_00_name_str \}
  \}
}
\end{verbatim}

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

\begin{verbatim}
\str_if_empty:NTF \l@@_name_str
{\cs_if_exist:cT \{@@_last_col_\int_use:N \g_@@_env_int
  \int_set:Nn \l@@_last_col_int
  \use:c \{@@_last_col_\int_use:N \g_@@_env_int \}
  \}
}
\str_if_empty:NTF \l@@_name_str
{\cs_if_exist:cT \{@@_last_col_\l_00_name_str
  \int_set:Nn \l@@_last_col_int
  \use:c \{@@_last_col_\l_00_name_str \}
  \}
}
\end{verbatim}

A value of $-1$ for the counter $\texttt{l@@_last_col_int}$ means that the user has used the option \texttt{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).

\begin{verbatim}
\int_compare:nNnT \l@@_last_col_int = {-1}
{\str_if_empty:NTF \l@@_name_str
{\cs_if_exist:cT \{@@_last_col_\int_use:N \g_@@_env_int
  \int_set:Nn \l@@_last_col_int
  \use:c \{@@_last_col_\int_use:N \g_@@_env_int \}
  \}
}
{\cs_if_exist:cT \{@@_last_col_\l_00_name_str
  \int_set:Nn \l@@_last_col_int
  \use:c \{@@_last_col_\l_00_name_str \}
  \}
}
\end{verbatim}

The code in $\texttt{@@_pre_array}$: is used only by \{\texttt{NiceArrayWithDelims}\}.

$\texttt{@@_pre_array}$:

We compute the width of the two delimiters.

\begin{verbatim}
\dim_zero_new:N \l_@@_left_delim_dim
\dim_zero_new:N \l_@@_right_delim_dim
\end{verbatim}
The command \bBigg@ is a command of amsmath.
\hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #1 $ }
\dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
\hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #2 $ }
\dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }

The array will be composed in a box (named \l_@@_the_array_box) because we have to do manipulations concerning the potential exterior rows.
\box_clear_new:N \l_@@_the_array_box

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).

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:NT \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_@@_right_margin_dim
\skip_horizontal:N \l_@@_extra_right_margin_dim
\hbox_set_end:

End of the construction of the array (in the box \l_@@_the_array_box).

It the user has used the key 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).
\int_compare:nNnT \l_@@_last_row_int > { -2 }
{ \bool_if:NT \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
  }
}
Now, the definition of $\texttt{\textbackslash c@jCol}$ and $\texttt{\textbackslash g_\@@_col_total_int}$ change: $\texttt{\textbackslash c@jCol}$ will be the number of columns without the “last column”; $\texttt{\textbackslash g_\@@_col_total_int}$ will be the number of columns with this “last column”.

\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 \}
\}
\bool_if:TF \l_@@_Matrix_bool
\{ \int_compare:nNnT \l_@@_last_col_int > { -1 } \}
\{ \@@_error:n \{ \text{last\-col\-not\-used} \} \}
\}
\int_gset_eq:NN \g_\@@_row_total_int \c@iRow
\int_compare:nNnT \l_@@_last_row_int > { -1 } \{ \int_gdecr:N \c@iRow \}
\}
\end{verbatim}

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

\begin{verbatim}
\int_gset_eq:NN \g_\@@_row_total_int \c@iRow
\int_compare:nNnT \l_@@_last_row_int > { -1 } \{ \int_gdecr:N \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 $\texttt{\textbackslash g_\@@_width_first_col_dim}$: see p. 90).

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

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

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

Now, in the case of an environment $\{pNiceArray\}$, $\{bNiceArray\}$, etc. We compute $\l_\texttt{tmpa_dim}$ which is the total height of the “first row” above the array (when the key \texttt{first\-row} is used).

\begin{verbatim}
\int_compare:nNnT \l_@@_first_row_int = 0
\{ \dim_set_eq:NN \l_\texttt{tmpa_dim} \g_\@@_dp_row_zero_dim
\dim_add:Nn \l_\texttt{tmpa_dim} \g_\@@_ht_row_zero_dim \}
\end{verbatim}

We compute $\l_\texttt{tmpb_dim}$ 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 $\l_\texttt{tmpb_dim}$ means that there is no “last row.”

\begin{verbatim}
\int_compare:nNnT \l_@@_last_row_int > { -2 } \{ \dim_set_eq:NN \l_\texttt{tmpb_dim} \g_\@@_ht_last_row_dim
\dim_add:Nn \l_\texttt{tmpb_dim} \g_\@@_dp_last_row_dim \}
\end{verbatim}

\begin{verbatim}
\hbox_set:Nn \l_\texttt{tmpa_box}
\end{verbatim}

---

\textsuperscript{38}We remind that the potential “first column” (exterior) has the number 0.

\textsuperscript{39}A value of $-1$ for $\l_\texttt{tmpb_dim}$ means that there is a “last row” but the the user have not set the value with the option \texttt{last row} (and we are in the first compilation).
We take into account the “first row” (we have previously computed its total height in \l_tmpa_dim).
The \hbox:N (or \ 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 }
\bool_if:NTF \l_@@_NiceTabular_bool
{ \skip_horizontal:N -\arraycolsep }
\@@_use_arraybox_with_notes_c:
\bool_if:NTF \l_@@_NiceTabular_bool
{ \skip_horizontal:N -\tabcolsep }
\bool_if:NTF \l_@@_NiceTabular_bool
{ \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
}

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 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. 91).
\bool_if:NT \g_@@_last_col_found_bool
{ \skip_horizontal:N \g_@@_width_last_col_dim
\skip_horizontal:N \arraycolsep }
\@@_after_array:
\egroup
\bool_if:NT \c_@@_footnote_bool \endsavenotes
}

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

We construct the preamble of the array

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

The argument of \@@_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 \g_@@_preamble_tl.
\cs_new_protected:Npn \@@_construct_preamble:n #1
\{ First, we will do an “expansion” of the preamble with the tools of the package \texttt{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 \texttt{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 protections of the letters \texttt{w} and \texttt{W}. We don’t want these column types 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 \texttt{\{tabular\}} of \texttt{array} in some cells of our array. That’s why we do those redefinitions in a \TeX{} group.

```latex
\begin{group}
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 } \}
  \{
    \@@_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). \texttt{\temptokena} \texttt{\{ \#1 \}} Initialisation of a flag used by \texttt{array} to detect the end of the expansion. \texttt{\tempswatrue} The following line actually does the expansion (it’s has been copied from \texttt{array.sty}). \texttt{\whiledo { \iftempsw } \fi { \tempswafalse \the \NC@list }}

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}.

```latex
\begin{verbatim}
\int_gzero_new:N \c@jCol
\bool_if:NTF \l_@@_vlines_bool
  \{ \tl_gset:Nn \g_@@_preamble_tl
    { ! { \skip_horizontal:N \arrayrulewidth } }
  \}
\tl_gclear:N \g_@@_preamble_tl
\int_zero:N \l_tmpa_int
\end{verbatim}
```

Now, we actually patch the preamble (and it is constructed in \texttt{\g_@@_preamble_tl}).

```latex
\begin{verbatim}
\int_zero:N \l_tmpa_int
\exp_after:wN \@@_patch_preamble:n \temptokena \q_stop
\tl_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
\end{verbatim}
```

Now, we replace \texttt{\columncolor} by \texttt{\@@_columncolor_preamble}. \texttt{\bool_if:NT \l_@@_colortbl_like_bool}

```latex
\begin{verbatim}
\\regex_replace_all:NnN \c_@@_columncolor_regex
  { \c { \@@_columncolor_preamble } }
  \g_@@_preamble_tl
\end{verbatim}
```

We complete the preamble with the potential “exterior columns”.

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

78
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 \texttt{NiceTabular*} ($\l_\text{tabular\_width\_dim}=0pt$).

\begin{verbatim}
\nl_\text{last\_col_int} \gt \{-1\}
{ \tl_gput_right:Nn \g_\text{preamble\_tl} \{ c \} }
\int_compare:nNnTF \l_\text{last\_col_int} \gt \{-1\}
{ \tl_gput_right:NV \g_\text{preamble\_tl} \c_\text{preamble\_last\_col\_tl} }
\bool_lazy_all:nT
{ \l_\text{NiceArray_bool}
{ \bool_not_p:n \l_\text{NiceTabular_bool} }
{ \bool_not_p:n \l_\text{vlines_bool} }
{ \bool_not_p:n \l_\text{exterior\_arraycolsep_bool} }
}
{ \tl_gput_right:Nn \g_\text{preamble\_tl} \{ @ { } \} }
\dim_compare:nNnT \l_\text{tabular\_width\_dim} = \c_{\text{zero\_dim}}
{ \tl_gput_right:Nn \g_\text{preamble\_tl} \{ > \{ \text{\_error\_too\_much\_cols:} \} \ 1 \} }
\group_end:
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \l_\text{patch\_preamble:n} \#1
{ \str_case:nnF \#1
{ c { \l_\text{patch\_preamble_i:n} \#1 }
 l { \l_\text{patch\_preamble_i:n} \#1 }
 r { \l_\text{patch\_preamble_i:n} \#1 }
 > { \l_\text{patch\_preamble_ii:nn} \#1 }
 ! { \l_\text{patch\_preamble_ii:nn} \#1 }
 @ { \l_\text{patch\_preamble_ii:nn} \#1 }
 | { \l_\text{patch\_preamble_ii:nn} \#1 }
 p { \l_\text{patch\_preamble_iv:nnn t} \#1 }
 m { \l_\text{patch\_preamble_iv:nnn c} \#1 }
 b { \l_\text{patch\_preamble_iv:nnn b} \#1 }
 \l_\text{w:} { \l_\text{patch\_preamble_v:nnnn} \#1 }
 \l_\text{W:} { \l_\text{patch\_preamble_v:nnnn} \{ \cs_set_eq:NN \hss \hfil \} \#1 }
 \l_\text{true\_c:} { \l_\text{patch\_preamble_vii:n} \#1 }
 \q_stop { }
 }{ \str_if_eq:VnTF \l_\text{letter\_for\_dotted\_lines_str} \#1
{ \l_\text{patch\_preamble_vii:nn} \#1 }
{ \l_\text{fatal:nn} \{ \text{unknown\_column\_type} \} \#1 }
}
\end{verbatim}

For \texttt{c}, \texttt{l} and \texttt{r}

\begin{verbatim}
\cs_new_protected:Npn \l_\text{patch\_preamble_i:n} \#1
{ \tl_gput_right:Nn \g_\text{preamble\_tl} \{ > \\text{\_Cell:} \#1 < \text{\_end\_Cell:} \} }
\end{verbatim}
We increment the counter of columns.
\int_incr:N \c@jCol
\@@_patch_preamble_viii:n
}

For $>$, $!$ and $\circ$
\cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
\tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
\@@_patch_preamble:n
}

For $|$\l_tmpa_int is the number of successive occurrences of $|$\int_incr:N \l_tmpa_int
\@@_patch_preamble_iii_i:n #1
\cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
\str_if_eq:nnTF { #1 } \exp_not:N !
\tl_gput_right:Nx \g_@@_preamble_tl
\skip_horizontal:n
\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
\begin { minipage } \[ #1 \] { #3 }
\mode_leave_vertical:
\box_use:N \@arstrutbox
\end { minipage } \@@_end_Cell: }
\@@_Cell:
\begin { minipage } { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } { \int_zero:N \l_tmpa_int
\@@_patch_preamble:n #1
}
We increment the counter of columns.
\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n
}

For \texttt{w} and \texttt{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:
{c
< {\@@_end_Cell:
#1\hbox_set_end:\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 \texttt{\@@_true_c}: which will appear in our redefinition of the columns of type \texttt{S} (of \texttt{siunitx}).
\cs_new_protected:Npn \@@_patch_preamble_vi:n #1
{\tl_gput_right:Nn \g_@@_preamble_tl { c }
 We increment the counter of columns.
\int_gincr:N \c@jCol
\@@_patch_preamble_viii:n
}
\cs_new_protected:Npn \@@_patch_preamble_vii:n #1
{\tl_gput_right:Nn \g_@@_preamble_tl { ! { \skip_horizontal:N \arrayrulewidth } }
\@@_patch_preamble:n { #1 }
}

The command \texttt{\@@_vdottedline:n} is protected, and, therefore, won’t be expanded before writing on \texttt{\g_@@_internal_code_after_tl}.
\tl_gput_right:Nx \g_@@_internal_code_after_tl
{\@@_vdottedline:n { \int_use:N \c@jCol } }
\@@_patch_preamble:n
}

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

The command \texttt{\@\_put\_box\_in\_flow:} puts the box \texttt{\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 \texttt{\l\_tmpa\_dim} and the total height of the potential last row in \texttt{\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 \texttt{\@\_put\_box\_in\_flow\_i:} is used when the value of \texttt{\l\_@@\_baseline\_str} is different of \texttt{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 \pgf@y 
\@@_qpoint:n { row - \@@_succ:n \c@iRow } 
\dim_gadd:Nn \g_tmpa_dim \pgf@y 
\dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim } 
\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 - base } 
\str_if:VnF \l_@@_baseline_str { t } { \int_set:Nn \l_tmpa_int 1 } 
{ \int_set_eq:NN \l_tmpa_int \c@iRow } 
\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 } 
\dim_gsub:Nn \g_tmpa_dim \pgf@y 
\end{verbatim}

Now, \texttt{\g\_tmpa\_dim} contains the \texttt{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 } 
\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 } 
\dim_gsub:Nn \g\_tmpa\_dim \pgf@y 
\end{verbatim}

We take into account the position of the mathematical axis.
Now, $g_{\text{tmpa\_dim}}$ contains the value of the $y$ translation we have to to.

The \TeX group is for potential specifications in the $l_{\text{@\_notes\_code\_before\_tl}}$.

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

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

The two dimensions $\texttt{\aboverulesep}$ et $\texttt{\heavyrulewidth}$ are parameters defined by booktabs.

$\texttt{\CT@arc@}$ is the specification of color defined by color\texttt{bl} but you use it even if color\texttt{bl} is not loaded.

The case of \texttt{baseline} equal to \texttt{b}. Remember that, when the key \texttt{b} is used, the \texttt{\{array\}} (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}.

The case of \texttt{baseline} equal to \texttt{b}. Remember that, when the key \texttt{b} is used, the \texttt{\{array\}} (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}.
\@_qpoint:n \{row - 1\}
\dim_gset_eq:NN \g_tmpa_dim \pgf@y
\@_qpoint:n \{row - \int_use:N \c@iRow - base\}
\dim_gsub:Nn \g_tmpa_dim \pgf@y
@endpgfpicture
\dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
\int_compare:nNnT \l_@@_first_row_int = 0
{ \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
  \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
}
\box_move_up:nn \g_tmpa_dim \{@@_use_arraybox_with_notes_c:\}
}

Now, the general case (hence the g in the name).
\cs_new_protected:Npn \@@_use_arraybox_with_notes:
{\@_use_arraybox_with_notes_c:}

We convert a value of \(t\) to a value of \(1\).
\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 \(\l_@@_int\).
\int_set:Nn \l_@@_int \l_@@_baseline_str
\bool_lazy_or:nnT
{\int_compare_p:nNn \l_@@_int < \l_@@_first_row_int}
{\int_compare_p:nNn \l_@@_int > \g_@@_row_total_int}
{ \@@_error:n \{bad-value-for-baseline\}
  \int_set:Nn \l_@@_int 1
}
\pgfpicture
\@_qpoint:n \{row - 1\}
\dim_gset_eq:NN \g_tmpa_dim \pgf@y
\@_qpoint:n \{row - \int_use:N \l_@@_int - base\}
\dim_gsub:Nn \g_tmpa_dim \pgf@y
@endpgfpicture
\dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
\int_compare:nNnT \l_@@_first_row_int = 0
{ \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
  \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
}
\box_move_up:nn \g_tmpa_dim \{@@_use_arraybox_with_notes_c:\}
}

The command \@@_put_box_in_flow_bis: is used when the option 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.
\cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
{\@@_put_box_in_flow_bis:nn\#1\#2}

We will compute the real width of both delimiters used.
\dim_zero_new:N \l_@@_real_left_delim_dim
\dim_zero_new:N \l_@@_real_right_delim_dim
\bbox_set:Nn \l_tmpb_box
{\c_math_toggle_token \left \#1 \vcenter
{\bbox_to_ht:nn \{\bbox_ht:N \l_tmpa_box + \bbox_dp:N \l_tmpa_box\}}
\bbox_to_dp:nn \{\bbox_dp:N \l_tmpa_box + \bbox_ht:N \l_tmpa_box\}

Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.

\begin{NiceArrayWithDelims}
\end{NiceArrayWithDelims}

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).

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}).

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}).
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 catched 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.

\@@_light_syntax_i #1 \CodeAfter \q_stop

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.

\cs_new_protected:Npn \@@_light_syntax_i #1 \CodeAfter #2 \q_stop
{ }
\tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

The body of the array, which is stored in the argument \#1, is now splitted into items (and not tokens).
\seq_gclear_new:N \g_@@_rows_seq
\tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
\exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

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.

\int_compare:nNnT \l_@@_last_row_int = { -1 }
{ \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }

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

We need a global affectation because, when executing \l_tmpa_tl, we will exit the first cell of the array.
\seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
\exp_args:HV \@@_line_with_light_syntax_i:n \l_tmpa_tl
\seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
\@@_create_col_nodes:
\endarray

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).
\cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
{ \str_if_eq:VnT \g_@@_name_env_str { #2 } { \\ } }
\cs_new_protected:Npn \@@_line_with_light_syntax:n #1
{ \tl_if_empty:nF { #1 } { \\ \@@_line_with_light_syntax_i:n { #1 } } }
\cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
{ \seq_gclear_new:N \g_@@_cells_seq
\seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
\seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
\exp_args:NNV \@@_line_with_light_syntax_i:n \l_tmpa_tl
\seq_map_inline:Nn \g_@@_cells_seq { & ##1 }

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).
\cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
{ \str_if_eq:VnT \g_@@_name_env_str { #2 } { \\ } }
\cs_new_protected:Npn \@@_fatal:n { empty-environment }

86
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).

\cs_new:Npn \@@_create_col_nodes: 
{ \crcr \int_compare:nNnT \l_@@_first_col_int = 0 
{ \omit \skip_horizontal:N -2\col@sep \bool_if:NT \l_@@_code_before_bool 
{ \pgfsys@markposition { \@@_env: - col - 0 } } \pgfpicture \pgfrememberpicturepositiononpagetrue \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin \str_if_empty:NF \l_@@_name_str 
{ \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } } \endpgfpicture & 
\omit \bool_gset_true:N \g_@@_row_of_col_done_bool \omit \int_compare:nNnTF \l_@@_first_col_int = 0 
{ \bool_if:NT \l_@@_code_before_bool 
{ \hbox 
{ \skip_horizontal:N -0.5\arrayrulewidth \pgfsys@markposition { \@@_env: - col - 1 } \pgfcoordinate { \@@_env: - col - 1 } \pgfpoint { -0.5 \arrayrulewidth } \c_zero_dim } \str_if_empty:NF \l_@@_name_str 
{ \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } } \endpgfpicture 
} 
{ \bool_if:NT \l_@@_code_before_bool 
{ \hbox 
{ \skip_horizontal:N 0.5 \arrayrulewidth \pgfsys@markposition { \@@_env: - col - 1 } \pgfcoordinate { \@@_env: - col - 1 } \str_if_empty:NF \l_@@_name_str 
{ \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } } \endpgfpicture 
} 
{ \bool_if:NT \l_@@_code_before_bool 
{ \hbox 
{ \skip_horizontal:N 0.5 \arrayrulewidth \pgfsys@markposition { \@@_env: - col - 1 } \pgfcoordinate { \@@_env: - col - 1 } \sk
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.

\[ \texttt{\skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } } \]

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.

\[ \texttt{\int_gset:Nn \g_tmpa_int 1 } \]

The incrementation of the counter $\texttt{\g_tmpa_int}$ must be done after the $\texttt{\omit}$ of the cell.

\[ \texttt{\int_gincr:N \g_tmpa_int } \]

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.

\[ \texttt{\int_gset:Nn \g_tmpa_int 1 } \]

The incrementation of the counter $\texttt{\g_tmpa_int}$ must be done after the $\texttt{\omit}$ of the cell.
We create the \texttt{col} node on the right of the current column.

```latex
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
{ \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
\str_if_empty:NF \l_@@_name_str
\{ \l_@@_name_str - col - \@@_succ:n \g_tmpa_int \}
\{ \@@_env: - col - \@@_succ:n \g_tmpa_int \}
\endpgfpicture
\bool_if:NT \g_@@_last_col_found_bool
{ \bool_if:NT \l_@@_code_before_bool
  { \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
\skip_horizontal:N 2\col@sep
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
\pgfpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
\str_if_empty:NF \l_@@_name_str
\{ \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int \}
\{ \@@_env: - col - \@@_succ:n \g_@@_col_total_int \}
\endpgfpicture
\skip_horizontal:N -2\col@sep
\cr
}
```

Here is the preamble for the “first column” (if the user uses the key \texttt{first-col})

```latex
\tl_const:Nn \c_@@_preamble_first_col_tl
{ > }
\@@_begin_of_row:
The contents of the cell is constructed in the box \texttt{\l_@@_cell_box} because we have to compute some dimensions of this box.

```latex
\hbox_set:Nw \l_@@_cell_box
\@@_math_toggle_token:
\bool_if:NT \l_@@_small_bool \scriptstyle
We insert \texttt{\l_@@_code_for_first_col_tl}... but we don’t insert it in the potential “first row” and in the potential “last row”:

```latex
\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 }
```
Be careful: despite this letter \l the cells of the “first column” are composed in a \hboxoverlap_left:n manner since they are composed in a \hboxoverlap_left:n.

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

The content of the cell is inserted in an overlapping position.

Here is the preamble for the “last column” (if the user uses the key last-col).

With the flag \g_@@_last_col_found_bool, we will know that the “last column” is really used.

The contents of the cell is constructed in the box \l_tmpa_box because we have to compute some dimensions of this box.

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”.

\l_@@_code_for_first_col_tl
\xglobal \colorlet { nicematrix-first-col } { . }
}\}

\l_@@_code_for_last_col_tl
\xglobal \colorlet { nicematrix-last-col } { . }
}
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 { #1 NiceArray } }
\@@_test_if_math_mode:
\NiceArrayWithDelims #2 #3
\}
\{ \endNiceArrayWithDelims \}
\}
\@@_def_env:nnn p ( )
\@@_def_env:nnn b [ ]
\@@_def_env:nnn B \{ \}
\@@_def_env:nnn v | |
\@@_def_env:nnn V \| \|
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 }
\list_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 }

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 }
\endNiceArray
\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 }
\endNiceArray

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 \{NiceArray\}, \{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 \l_@@_last_col_int in that case.
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{\_\_\_\_last_col_int}.

If the key \texttt{code-before} is used, we have to write on the \texttt{aux} file the actual size of the array.

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 option \texttt{light-syntax} is used, we have nothing to write since, in this case, the number of rows is directly determined.

It's also time to give to \texttt{\_\_\_\_last_row_int} its real value. But, if the user had used the option \texttt{last-row} without value, we write in the \texttt{aux} file the number of that last row for the next run.

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 user has used a key \texttt{last-row} in an environment with preamble (like \texttt{pNiceArray}) and that last row has not been found, we have to increment the value because it will be decreased when used in the \texttt{code-before}.

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

We write also the potential content of \texttt{g_@@_pos_of_blocks_seq} (it will be useful if the commands \texttt{rowcolors} is used with the key \texttt{respect-blocks}.

\begin{verbatim}
\seq_gset_from_clist:cn
\{
\int_use:N \g_@@_env_int \seq \}
\\texttt{\rowcolors} is used with the key \texttt{respect-blocks}.
\end{verbatim}

By default, the diagonal lines will be parallelized\textsuperscript{40}. There are two types of diagonals lines: the \texttt{Ddots} diagonals and the \texttt{Iddots} diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current \texttt{NiceArray} environment.

\begin{verbatim}
\bool_if:NT \l_@@_parallelize_diags_bool
\{
\int_gzero_new:N \g_@@_ddots_int
\int_gzero_new:N \g_@@_iddots_int
\}
\end{verbatim}

The dimensions \texttt{g_@@_delta_x_one_dim} and \texttt{g_@@_delta_y_one_dim} will contain the $\Delta_x$ and $\Delta_y$ of the first \texttt{Ddots} diagonal. We have to store these values in order to draw the others \texttt{Ddots} diagonals parallel to the first one. Similarly \texttt{g_@@_delta_x_two_dim} and \texttt{g_@@_delta_y_two_dim} are the $\Delta_x$ and $\Delta_y$ of the first \texttt{Iddots} diagonal.

\begin{verbatim}
\dim_gzero_new:N \g_@@_delta_x_one_dim
\dim_gzero_new:N \g_@@_delta_y_one_dim
\dim_gzero_new:N \g_@@_delta_x_two_dim
\dim_gzero_new:N \g_@@_delta_y_two_dim
\}
\end{verbatim}

If the option \texttt{small} is used, the values \texttt{\l_@@_radius_dim} and \texttt{\l_@@_inter_dots_dim} (used to draw the dotted lines created by \texttt{hdottedline} and \texttt{vdotteline} and also for all the other dotted lines when \texttt{line-style} is equal to \texttt{standard}, which is the initial value) are changed.

\begin{verbatim}
\bool_if:NT \l_@@_small_bool
\{
\dim_set:Nn \l_@@_radius_dim { 0.37 pt }
\dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }
\}
\end{verbatim}

The dimension \texttt{\l_@@_xdots_shorten_dim} corresponds to the option \texttt{xdots/shorten} available to the user. That’s why we give a new value according to the current value, and not an absolute value.

\textsuperscript{40}It’s possible to use the option \texttt{parallelize-diags} to disable this parallelization.
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.

\begin{verbatim}
\@@_compute_corners:
\end{verbatim}

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.

\begin{verbatim}
\bool_lazy_all:nT
{
{ \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
{ \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
{ \seq_if_empty_p:N \l_@@_empty_corner_cells_seq }
}
{ \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
\cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn }
\bool_if:NT \l_@@_hlines_bool \@@_draw_hlines:
\bool_if:NT \l_@@_vlines_bool \@@_draw_vlines:
\g_@@_internal_code_after_tl
\tl_gclear:N \g_@@_internal_code_after_tl
\end{verbatim}

We draw the blocks. We have to revert to a clean version of \ialign because there may be tabulars in the \Block instructions that will be composed now.

\begin{verbatim}
\cs_set_eq:NN \ialign \@@_old_ialign:
\seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
\end{verbatim}

Now, the code-after.

\begin{verbatim}
\bool_if:NT \c_@@_tikz_loaded_bool
{\tikzset
{ every-picture /.style =
{ overlay ,
  remember-picture ,
  name-prefix = \@@_env: -
}
}
\cs_set_eq:NN \line \@@_line
\end{verbatim}

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 \Code-after to be no-op now.

\begin{verbatim}
\cs_set_eq:NN \CodeAfter \prg_do_nothing:
\end{verbatim}

And here’s the code-after:

\begin{verbatim}
\g_nicematrix_code_after_tl
\tl_gclear:N \g_nicematrix_code_after_tl
\group_end:
\end{verbatim}

\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 colortbl. 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
\iow_now:Nx \@mainaux
\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\footnote{\eg \color\{rgb\}{0.5,0.5,0}). This command is used by \CT@arc@ but we use it also for compatibility with colortbl. But we want also to be able to use color for the rules of the array when colortbl is not loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by colortbl.

\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:
\{ \c_@@_pgfortikzpicture_tl \@@_draw_dotted_lines_i:
\c_@@_endpgfortikzpicture_tl \}
\}

The following command \textit{must} be protected because it will appear in the construction of the command \@@_draw_dotted_lines::

\cs_new_protected:Npn \@@_draw_dotted_lines_i:
\{ \pgfrememberpositiononpagetrue \pgfrelevantforpicturesizesfalse \g_@@_HVdotsfor_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 } \}

\footnote{\eg \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.

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

The command \texttt{\textbackslash@@\_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:

- \texttt{\l@@\_initial\_i\_int} and \texttt{\l@@\_initial\_j\_int} which are the coordinates of one extremity of the line;
- \texttt{\l@@\_final\_i\_int} and \texttt{\l@@\_final\_j\_int} which are the coordinates of the other extremity of the line;
- \texttt{\l@@\_initial\_open\_bool} and \texttt{\l@@\_final\_open\_bool} to indicate whether the extremities are open or not.

\begin{verbatim}
2179 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4 
2180 { 
2181 First, we declare the current cell as “dotted” because we forbide intersections of dotted lines.
2182 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { } 
2183 Initialization of variables.
2184 \int_set:Nn \l_@@_initial_i_int { #1 } 
2185 \int_set:Nn \l_@@_initial_j_int { #2 } 
2186 \int_set:Nn \l_@@_final_i_int { #1 } 
2187 \int_set:Nn \l_@@_final_j_int { #2 } 
2188 We will do two loops: one when determinating the initial cell and the other when determinating the final cell. The boolean \texttt{\l@@\_stop\_loop\_bool} will be used to control these loops. In the first loop, we search the “final” extremity of the line.
2189 \bool_set_false:N \l_@@_stop_loop_bool 
2190 \bool_do_until:Nn \l_@@_stop_loop_bool 
2191 { 
2192 \int_add:Nn \l_@@_final_i_int { #3 } 
2193 \int_add:Nn \l_@@_final_j_int { #4 } 
2194 We test if we are still in the matrix.
2195 \bool_set_false:N \l_@@_final_open_bool 
2196 \int_compare:nNnTF \l_@@_final_i_int > \c@iRow 
2197 { \int_compare:nNnTF \l_@@_final_i_int > \c@iRow 
2198 { \int_compare:nNnTF \l_@@_final_j_int > \c@jCol } 
2199 } 
2200 } 
2201 { \int_compare:nNnTF \l_@@_final_j_int < 1 
2202 { 
2203 } 
2204 
\end{verbatim}
\int_compare:nNnT \c@jCol < \l_@@_final_j_int
\int_compare:nNnT \l_@@_final_j_int \int \l_@@_final_i_int
\cs_if_exist:cTF
\cs_set:cpn
{ @@_dotted_
\int_use:N \l_@@_final_i_int - \int_use:N \l_@@_final_j_int
} 
For \_\_initial\_i\_int and \_\_initial\_j\_int the programmation is similar to the previous one.

```latex
\bool_set_false:N \_\_stop\_loop\_bool
\bool_do_until:Nn \_\_stop\_loop\_bool
\{ 
  \int_sub:Nn \_\_initial\_i\_int { #3 }
  \int_sub:Nn \_\_initial\_j\_int { #4 }
  \bool_set_false:N \_\_initial\_open\_bool
  \int_compare:nNnTF \_\_initial\_i\_int < 1
  \{ 
    \int_compare:nNnTF \_\_initial\_j\_int < 1
    \{ 
      \int_compare:nNnTF \_\_initial\_j\_int = 1
      \{ \bool_set_true:N \_\_initial\_open\_bool \}
      \{ \bool_set_true:N \_\_initial\_open\_bool \}
    \}
    \int_compare:nNnTF \_\_initial\_j\_int > \c@jCol
    \{ 
      \int_compare:nNnT \_\_initial\_j\_int = { -1 }
      \{ \bool_set_true:N \_\_initial\_open\_bool \}
      \}
  \}
  \bool_if:NTF \_\_initial\_open\_bool
  \{ 
    \int_add:Nn \_\_initial\_i\_int { #3 }
    \int_add:Nn \_\_initial\_j\_int { #4 }
    \bool_set_true:N \_\_stop\_loop\_bool
  \}
  \{ 
    \cs_if_exist:cTF
    \{ \_\_ dotted \}
    \int_use:N \_\_initial\_i\_int - 
    \int_use:N \_\_initial\_j\_int
    \}
  \}
\bool_if:NTF \_\_initial\_open\_bool
\{ 
  \int_add:Nn \_\_initial\_i\_int { #3 }
  \int_add:Nn \_\_initial\_j\_int { #4 }
  \bool_set_true:N \_\_stop\_loop\_bool
\}
\{ 
  \cs_if_exist:cTF
  \{ \_\_ dotted \}
  \int_use:N \_\_initial\_i\_int - 
  \int_use:N \_\_initial\_j\_int
  \}
\}
```

---

99
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 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.
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.

\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Ldots:

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 \@@_draw_Cdots:nnn #1 #2 #3
{
\bool_if:NTF \l_@@_initial_open_bool
\{\@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
\dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
\dim_add:Nn \l_@@_x_initial_dim \@@_tab_or_array_colsep:
\@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int ~ base }
\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
\}
\bool_if:NTF \l_@@_final_open_bool
\{\@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
\dim_set_eq:NN \l_@@_x_final_dim \pgf@x
\dim_sub:Nn \l_@@_x_final_dim \@@_tab_or_array_colsep:
\@@_qpoint:n { row - \int_use:N \l_@@_final_i_int ~ base }
\dim_set_eq:NN \l_@@_y_final_dim \pgf@y
\}
\@@_draw_line:

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option line-style is used (?).

\dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
\@@_draw_line:

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.

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.

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.

\cs_new_protected:Npn \@@_actually_draw_Cdots:
{
  \bool_if:NTF \l_@@_initial_open_bool
  {
    \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_add:Nn \l_@@_x_initial_dim { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
  }
  \bool_if:NTF \l_@@_final_open_bool
  {
    \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_sub:Nn \l_@@_x_final_dim { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
  }
  \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
  {
    \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \dim_set_eq:NN \l_@@_y_initial_dim { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
    \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
    {
      \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
      \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
      \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
      \dim_set_eq:NN \l_@@_y_initial_dim { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
    }
  }
}
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.

The boolean \l_tmpa_bool indicates whether the column is of type 1 or may be considered as if.

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.

\begin{Verbatim}
\int_compare:nNnT \l_@@_last_col_int > \{-2 \}
\end{Verbatim}

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.

\begin{Verbatim}
\dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
\end{Verbatim}

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.

\begin{Verbatim}
\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
\end{Verbatim}
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.

\begin{group}
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Ddots:
\end{group}

The command \texttt{\@@\_actually\_draw\_Ddots} has the following implicit arguments:

- \texttt{\l_@@\_initial\_i\_int}
- \texttt{\l_@@\_initial\_j\_int}
- \texttt{\l_@@\_initial\_open\_bool}
- \texttt{\l_@@\_final\_i\_int}
- \texttt{\l_@@\_final\_j\_int}
- \texttt{\l_@@\_final\_open\_bool}.

We have retrieved the coordinates in the usual way (they are stored in \texttt{\l_@@\_x\_initial\_dim}, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

\begin{verbatim}
\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\texttt{y} \\
  \@@\_qpoint:n \col - \int_use:N \l_@@\_initial\_j\_int \\
  \dim_set_eq:NN \l_@@\_x\_initial\_dim \pgf\texttt{x} \\
} \\
{ \@@\_set\_initial\_coords\_from\_anchor:n \south\texttt{e}ast }
\end{verbatim}

\begin{verbatim}
\bool_if:NTF \l_@@\_final\_open\_bool
{ \\
  \@@\_qpoint:n \row - \@@\_successor:n \l_@@\_final\_i\_int \\
  \dim_set_eq:NN \l_@@\_y\_final\_dim \pgf\texttt{y} \\
  \@@\_qpoint:n \col - \@@\_successor:n \l_@@\_final\_j\_int \\
  \dim_set_eq:NN \l_@@\_x\_final\_dim \pgf\texttt{x} \\
} \\
{ \@@\_set\_final\_coords\_from\_anchor:n \north\texttt{e}ast }
\end{verbatim}

We test if the diagonal line is the first one (the counter \texttt{\g_@@\_ddots\_int} is created for this usage).

\begin{verbatim}
\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 \\
}
\end{verbatim}
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 \texttt{\textbackslash 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 }
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_Iddots:
\group_end:
\}
\end{verbatim}

The command \texttt{\textbackslash @@_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
\} \{ \@@_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:

\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 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
{
  \pgfrememberpicturepositiononpagetrue
  \pgf@relevantforpicturesizefalse
  \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
    \@@_draw_standard_dotted_line:
    \@@_draw_non_standard_dotted_line:n:
}

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.
\draw
[
  #1 ,
  shorten-> = \l_@@_xdots_shorten_dim ,
  shorten<- = \l_@@_xdots_shorten_dim ,
]
  ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
-- node [ sloped , above ]
  { \scriptstyle \l_@@_xdots_up_tl }
node [ sloped , below ]
  { \scriptstyle \l_@@_xdots_down_tl }
  ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
\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
  \pgftransformshift
  { \pgfpointlineattime { 0.5 }
      { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
      { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
    }
  \pgftransformrotate
  { \fp_eval:n
      { atand ( \l_@@_y_final_dim - \l_@@_y_initial_dim ,
                  \l_@@_x_final_dim - \l_@@_x_initial_dim )
    }
  }
  \pgfnode
  { rectangle }
  { south }
  { \scriptstyle \l_@@_xdots_up_tl }
  \pgfnode
  { rectangle }
  { north }
  { \scriptstyle \l_@@_xdots_down_tl

108
The dimension $\l_@@_l_{\ dim}$ is the length $\ell$ of the line to draw. We use the floating point reals of \texttt{expl3} to compute this length.

\begin{verbatim}
\dim_zero_new:N \l_@@_l_dim
\dim_set:Nn \l_@@_l_dim {
  \fp_to_dim:n {
    \sqrt {
      ( \l_@@_x_{\ final}\dim - \l_@@_x_{\ initial}\dim)^2 
      +
      ( \l_@@_y_{\ final}\dim - \l_@@_y_{\ initial}\dim)^2
    }
  }
}
\end{verbatim}

It seems that, during the first compilations, the value of $\l_@@_l_{\ dim}$ 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 \texttt{aux} file to say that one more compilation should be done.

\begin{verbatim}
\bool_lazy_or:nnF
{ \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
{ \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
\@@_draw_standard_dotted_line_i:
\end{verbatim}

The integer $\l_{\ tmpa}_\int$ is the number of dots of the dotted line.

\begin{verbatim}
\bool_if:NTF \l_@@_initial\_open\_bool
{ \bool_if:NTF \l_@@_final\_open\_bool
  { \int_set:Nn \l_{\ tmpa}_\int \l_@@_l_dim \l_@@_inter\_dots\_dim }
}
\end{verbatim}
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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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.

In the loop, 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 arg spec of these commands. However, we don’t know the future catcode of \_ in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscores activates \_). That’s why these commands will be defined in a \AtBeginDocument and the arg spec will be rescanned.
\exp_args:NNV \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:NNV \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
}
}
\exp_args:NNV \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:NNV \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 }
\begin{verbatim}
  \draw-first .bool_set:N = \l_@@_draw_first_bool ,
  \draw-first .default:n = true ,
  \draw-first .value_forbidden:n = true
\end{verbatim}

The command \@@_Hspace: will be linked to \hspace in \{NiceArray\}.

\cs_new_protected:Npn \@@_Hspace: 
\begin{verbatim}
\bool_gset_true:N \g_@@_empty_cell_bool 
\hspace
\end{verbatim}

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{verbatim}
% \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 is mandatory.
\peek_remove_spaces:n 
\begin{verbatim}
\int_compare:nNnT #1 > 1 
\seq_gput_left:Nx \g_@@_multicolumn_cells_seq \int_use:N \c@iRow - \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 \int_use:N \c@jCol 
\int_eval:n { \c@jCol + #1 - 1 }
\end{verbatim}
\end{verbatim}

The \peek_remove_spaces:n is mandatory.

\peek_remove_spaces:n 
\begin{verbatim}
\int_compare:nNnT #1 > 1 
\seq_gput_left:Nx \g_@@_multicolumn_cells_seq
\\int_use:N \c@iRow - \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 \\int_use:N \c@jCol 
\int_eval:n { \c@jCol + #1 - 1 } 
\end{verbatim}

113
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`.

```
\cs_new:Npn \@@_Hdotsfor:
{\int_compare:nNnTF \c@jCol = 0
{ \@@_error:n { Hdotsfor~in~col~0 } }
{ \multicolumn { 1 } { c } { } \@@_Hdotsfor_i }
}
```

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
{\tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } }
\tl_set_rescan:Nno \l_@@_argspec_tl \l_@@_argspec_tl
\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
{ \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
{ \@@_Hdotsfor:nnnn \int_use:N \c@iRow \int_use:N \c@jCol \#2 \#3 \down = \exp_not:n \#4 , up = \exp_not:n \#5 }
\prg_replicate:nn { \#2 - 1 } { & \multicolumn { 1 } { c } { } }
}
\exp_after:wN \end_pgfkeys
\end_pgfkeys
```

Enf of `\AtBeginDocument`.

```
\cs_new_protected:Npn \@@_Hdotsfor:nnnn \#1 \#2 \#3 \#4
{ \bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
\int_set:Nn \l_@@_initial_i_int \#1
\int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
\int_compare:nNnTF \#2 = 1
{ \int_set:Nn \l_@@_initial_j_int \#2
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_final_open_bool }
\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 \#2
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_final_open_bool }
```

114
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:nmmn). This declaration is done by defining a special control sequence (to nil).

\int_step_inline:nnn { #2 } { #2 + #3 - 1 } { \cs_set:cpn { @@_dotted _ #1 - ##1 } { } }
\AtBeginDocument
{ \tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } } \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl { \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl { \@@_Vdotsfor:nnnn { \int_use:N \c@iRow } { \int_use:N \c@jCol } { #2 } { \color { \l_@@_xdots_color_tl } } } \@@_actually_draw_Ldots: \group_end: }
#1, #3,
down = \exp_not:n { #4 }, up = \exp_not:n { #5 }
}
}
}
}

End of \AtBeginDocument.
\cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
{
  \bool_set_false:N \l_@@_initial_open_bool
  \bool_set_false:N \l_@@_final_open_bool

  For the column, it’s easy.
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

  For the row, it’s a bit more complicated.
  \int_compare:nNnTF #1 = 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 { #1 - 1 }
    - \int_use:N \l_@@_initial_j_int
    }\int_set:Nn \l_@@_initial_i_int { #1 - 1 }
    {
    \cs_if_exist:cTF
      { pgf @ sh @ ns @ \@@_env:
      - \int_eval:n { #1 + #3 - 1 }
      - \int_use:N \l_@@_final_j_int
      }\int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
      {
      \int_compare:nNnTF { #2 } = \c@iRow
      \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
      \bool_set_true:N \l_@@_final_open_bool
      }\cs_if_exist:cTF
        { pgf @ sh @ ns @ \@@_env:
        - \int_eval:n { #1 + #3 }
        - \int_use:N \l_@@_final_j_int
        }\int_set:Nn \l_@@_final_i_int { #1 + #3 }
        {
        \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
        }
      }
    }\group_begin:
  \int_compare:nNnTF #1 + #3 - 1 = \c@iRow
  \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
  \bool_set_true:N \l_@@_final_open_bool
  { \cs_if_exist:cTF
    { pgf @ sh @ ns @ \@@_env:
    - \int_eval:n { #1 + #3 }
    - \int_use:N \l_@@_final_j_int
    }\int_set:Nn \l_@@_final_i_int { #1 + #3 }
    {
    \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
    \bool_set_true:N \l_@@_final_open_bool
    }
  }
  }
  { \group_end:
  \int_compare:nNnTF #2 = 0
  { \color { nicematrix-first-col }
  { \int_compare:nNnT { #2 } = \g_@@_col_total_int
    { \color { nicematrix-last-col }
    { \keys_set:n { NiceMatrix / xdots } { #4 }
      \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl }
        \@@_actually_draw_Vdots:
    }\@@_actually_draw_Vdots:
We declare all the cells concerned by the \dotsfor 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\}. The command will exit three levels of groups (only two in \{NiceTabular\} because there is not the group of the math mode to exit) in order to execute the command "\box_rotate:Nn \l_@@_cell_box { 90 }" just after the construction of the box \l_@@_cell_box.

\cs_new_protected:Npn \@@_rotate:
{ \bool_if:NTF \l_@@_NiceTabular_bool
{ \group_insert_after:N \@@_rotate_ii: }
{ \group_insert_after:N \@@_rotate_i: }
}
\cs_new_protected:Npn \@@_rotate_i: { \group_insert_after:N \@@_rotate_ii: }
\cs_new_protected:Npn \@@_rotate_ii: { \group_insert_after:N \@@_rotate_iii: }
\cs_new_protected:Npn \@@_rotate_iii:
{ \box_rotate:Nn \l_@@_cell_box { 90 } }

If we are in the last row, we want all the boxes composed with the command \rotate aligned upwards.

\int_compare:nNnT \c@iRow = \l_@@_last_row_int
{ \vbox_set_top:Nn \l_@@_cell_box
\vbox_to_zero:n { }
0.8 ex will be the distance between the principal part of the array and our element (which is composed with \rotate).
\skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
\box_use:N \l_@@_cell_box
}

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).\[42\]

\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

\[42\]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:nn`.

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).
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`.

```latex
\cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
\tl_set:Nn \l_tmpa_tl { #1 }
\tl_set:Nn \l_tmpb_tl { #2 }
\}
\NewDocumentCommand \@@_rowcolor { O { } m m }
\tl_if_blank:nF { #2 }
\{ \pgfpicture
\pgf@relevantforpicturesizefalse
\tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
\l_tmpa_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`.

```latex
\NewDocumentCommand \@@_columncolor { O { } m m }
\tl_if_blank:nF { #2 }
\{ \pgfpicture
\pgf@relevantforpicturesizefalse
\tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
```

Here an example:

```latex
\@@_rowcolor {red!15} {1,3,5-7,10-}
```

Here an example:

```latex
\@@_columncolor {red!15} {1,3,5-7,10-}
```
\l_tmpa_dim is the y-value of the top of the columns et \l_tmpb_dim is the y-value of the bottom.

\dim_set:Nn \l_tmpa_dim {\pgf@y + 0.5 \arrayrulewidth }
\@@_qpoint:n { row - \@@_succ:n \c@iRow }
\dim_set:Nn \l_tmpb_dim {\pgf@y + 0.5 \arrayrulewidth }
\clist_map_inline:nn { #3 }
{
\tl_set:Nn \l_tmpa_tl { ##1 }
\tl_if_in:NnTF \l_tmpa_tl { - } \{ \@@_cut_on_hyphen:w ##1 \q_stop \}
\tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
\tl_if_empty:NT \l_tmpb_tl { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
\int_compare:nNnT \l_tmpb_tl > \c@jCol
{ \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
\tl_if_in:NnTF \l_tmpa_tl { - } \{ \@@_cut_on_hyphen:w ##1 - ##1 \q_stop \}
\tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
\tl_if_empty:NT \l_tmpb_tl { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
\tl_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
\tl_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
\pgfpathrectanglecorners
{ \pgfpoint \l_tmpc_dim \l_tmpb_dim }
{ \pgfpoint \l_tmpd_dim \l_tmpa_dim }
} \pgfusepathqfill 
\endpgfpicture

Now, the numbers of both columns are in \l_tmpa_tl and \l_tmpb_tl.

\dim_set:Nn \l_@@_first_col_int { \pgf@x - 0.5 \arrayrulewidth }
\dim_set:Nn \l_@@_first_col_int { \pgf@x + 0.5 \arrayrulewidth }
\pgfpathrectanglecorners
{ \pgfpoint \l_tmpc_dim \l_tmpb_dim }
{ \pgfpoint \l_tmpd_dim \l_tmpa_dim }
} \pgfusepathqfill 
\endpgfpicture

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 
\pgfpicture
\pgfpicture
\pgfpicture
\pgfpathrectanglecorners
{ \pgfpoint \l_tmpc_dim \l_tmpb_dim }
{ \pgfpoint \l_tmpd_dim \l_tmpa_dim }
} \pgfusepathqfill 
\endpgfpicture

120
Here an example:

\begin{pgfpicture}
\pgf@relevantforpicturesizefalse
\tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
\@@_cut_on_hyphen:w #3 \q_stop
\bool_lazy_and:nnT
{ \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
{ \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
{ \@@_qpoint:n { row - \l_tmpa_tl } }
\dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
\@@_qpoint:n { col - \l_tmpb_tl }
\int_compare:nnTF \l_@@_first_col_int = \l_tmpb_tl
{ \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
{ \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
\@@_cut_on_hyphen:w #4 \q_stop
\bool_lazy_and:nnT \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 } }
\endpgfpicture

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:nnT \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 } }
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”.

The counter \l_tmpa_int will be the index of the loop.

The boolean \l_tmpa_bool will indicate whereas we are in a row of the first color or of the second color.

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.

We do until \l_tmpa_int > \c@iRow

We set \l_tmpb_int equal to \l_tmpa_int - \int_use:N \l_tmpb_int

\bool_set_false:N \l_tmpa_bool

We compare \l_tmpb_int (3) > \l_tmpb_int

\bool_lazy_or:nnTF

We return false

We return true

We set \l_tmpa_int equal to \l_tmpb_int + 1

We new protected: N \@@_rowcolors_i:nnnn #1 #2 #3 #4

We set \l_tmpb_seq equal to \\g_@@_env_int_seq

We compute in \l_tmpb_int the last row covered by a block.

We compare \l_tmpb_int (3) > \l_tmpb_int

We return false

We return true

We set \l_tmpa_int equal to \l_tmpb_int + 1
The following command return true when the block intersects the row \l_tmpa_int.

```latex
\prg_new_conditional:Nnn \@@_intersect_our_row:nnnn p
\bool_if:nTF
\int_compare_p:n { #1 \leq \l_tmpa_int }
&
\int_compare_p:n { \l_tmpa_int \leq #3 }
\prg_return_true:
\prg_return_false:
\NewDocumentCommand \@@_chessboardcolors { O { } m m }
\int_step_inline:nn { \int_use:N \c@iRow }
\int_step_inline:nn { \int_use:N \c@jCol }
\int_if_even:nTF { ####1 + ##1 }
{ \@@_cellcolor \[ #1 \] { #2 } }
{ \@@_cellcolor \[ #1 \] { #3 } }
{ ##1 - ####1 }
\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 }
\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 - \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
\columncolor \[ #1 \] { #2 } { \int_use:N \c@jCol }
}
```

When the user uses the key colortbl-like, the following command will be linked to \cellcolor in the tabular.

```latex
\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.

```latex
\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
```

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).

```latex
\tl_gput_left:Nx \g_nicematrix_code_before_tl
\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 `\newcolumntype` of `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 `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`). That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```latex
\cs_set_eq:NN \OnlyMainNiceMatrix \use:n
```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```latex
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
{\int_compare:nNnTF \l_@@_first_col_int = 0
{ \@@_OnlyMainNiceMatrix_i:n { #1 } }\int_compare:nNnTF \c@jCol = 0
{ \int_compare:nNnF \c@iRow = { -1 } { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }\int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }}\tl_set:Nx \l_tmpa_tl \tl_clear_new:N \l_tmpc_tl
```

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 a 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.

```latex
\cs_new_protected:Npn \@@_vline:nn #1 #2
{\l_tmpa_tl \int_compare:nNnT { #1 } < { \c@jCol + 2 } \begin{pgfpicture}
\@@_vline_i:nn { #1 } { #2 }
\endpgfpicture}
```

The following test is for the case where the user don’t use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```latex
\cs_new_protected:Npn \@@_vline_i:nn #1 #2
{\pgfmath \@@_vline_i:nn { #1 } { #2 } \endpgfpicture}
```

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.

```latex
\\l_tmpa_tl \l_tmpb_tl \l_tmpc_tl \tl_set:Nx \l_tmpb_tl { #1 } \tl_clear_new:N \l_tmpc_tl
```

\l_\text{tmpa_tl} is the number of row and \l_\text{tmpb_tl} the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_\text{tmpc_tl}.

```latex
\tl_set:Nx \l_\text{tmpb_tl} { #1 } \\tl_set:Nx \l_\text{tmpc_tl} \\tl_clear_new:N \l_\text{tmpc_tl}
```

The boolean \texttt{\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 \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 vertical 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_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 { #1 } { #2 } \l_tmpc_tl
    \tl_clear:N \l_tmpc_tl }
\tl_if_empty:NF \l_tmpc_tl
  { \@@_vline_ii:nnnn { #1 } { #2 } \l_tmpc_tl
    \tl_clear:N \l_tmpc_tl }
\end{verbatim}

We keep in memory that we have a rule to draw.

\begin{verbatim}
\seq_if_in:NxT \l_@@_empty_corner_cells_seq
  \l_tmpa_tl - \@@_pred:n \l_tmpb_tl
\bool_set_false:N \g_tmpa_bool
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_test_in_corner_v:
  \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
    \seq_if_in:NxT \l_@@_empty_corner_cells_seq
      \l_tmpa_tl - \@@_pred:n \l_tmpb_tl
      \bool_set_false:N \g_tmpa_bool
  \end{verbatim}

\begin{verbatim}
\int_compare:nNnTF \l_tmpb_tl = 1
  \seq_if_in:NxT \l_@@_empty_corner_cells_seq
    \l_tmpa_tl - \l_tmpb_tl
  \end{verbatim}

\begin{verbatim}
\int_compare:nNnTF \l_tmpb_tl = 1
  \seq_if_in:NxT \l_@@_empty_corner_cells_seq
    \l_tmpa_tl - \l_tmpb_tl
  \end{verbatim}
#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.

\cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
{\pgfrememberpicturepositiononpage\true\pgf@relevantforpicturesizefalse\@@_qpoint:n { row - #3 }\dim_set_eq:NN \l_tmpa_dim \pgf@y\@@_qpoint:n { col - #1 }\dim_set_eq:NN \l_tmpb_dim \pgf@x\@@_qpoint:n { row - \@@_succ:n { #4 } }\dim_set_eq:NN \l_tmpc_dim \pgf@y\bool_lazy_and:nnT\{ \int_compare_p:nNn { #2 } > 1 \}{ ! \tl_if_blank_p:V \CT@drsc@ }{ \group_begin: \CT@drsc@ \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }\dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }\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:"}
\@@_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:
{ \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. Wh, we have found a column corresponding to a rule to draw, we note its numver 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
      \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_tmpa_tl is the number of row and \l_tmpb_tl the number of column. Wh, we have found a column corresponding to a rule to draw, we note its numver 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
  \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
        \{
          \tl_if_empty:NT \l_tmpc_tl
            \{
              \\@@_hline_ii:nnnn
              \{
                \tl_set:Nn \l_tmpc_tl \l_tmpb_tl
                \{ #1 \}
                \{ #2 \}
                \{ \int_eval:n { \l_tmpb_tl - 1 } \}
                \tl_clear:N \l_tmpc_tl
              \}
            \}
        \}
    \}
\tl_if_empty:NF \l_tmpc_tl
  \{
    \\@@_hline_ii:nnnn
    \{
      \tl_set:Nn \l_tmpc_tl \l_tmpb_tl
      \{ #1 \}
      \{ #2 \}
      \{ \int_eval:n { \l_tmpb_tl - 1 } \}
      \tl_clear:N \l_tmpc_tl
    \}
  \}
\tl_if_empty:NF \l_tmpc_tl
  \{
    \\@@_hline_ii:nnnn
    \{
      \tl_set:Nn \l_tmpc_tl \l_tmpb_tl
      \{ #1 \}
      \{ #2 \}
      \{ \int_eval:n { \l_tmpb_tl - 1 } \}
      \tl_clear:N \l_tmpc_tl
    \}
  \}
\tl_if_empty:NF \l_tmpc_tl
  \{
    \\@@_hline_ii:nnnn
    \{
      \tl_set:Nn \l_tmpc_tl \l_tmpb_tl
      \{ #1 \}
      \{ #2 \}
      \{ \int_eval:n { \l_tmpb_tl - 1 } \}
      \tl_clear:N \l_tmpc_tl
    \}
  \}
\tl_if_empty:NF \l_tmpc_tl
  \{
    \\@@_hline_ii:nnnn
    \{
      \tl_set:Nn \l_tmpc_tl \l_tmpb_tl
      \{ #1 \}
      \{ #2 \}
      \{ \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:NxT \l_@@_empty_corner_cells_seq  
        \{ \@@_pred:n \l_tmpa_tl - \l_tmpb_tl  
          \bool_set_false:N \g_tmpa_bool \}  
      }  
      { \seq_if_in:NxT \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:NxT \l_@@_empty_corner_cells_seq  
            \{ \@@_pred:n \l_tmpa_tl - \l_tmpb_tl  
              \bool_set_false:N \g_tmpa_bool \}  
          }  
      }  
      { \bool_set_false:N \g_tmpa_bool \}  
  }  

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.
The command $@@_{\text{draw \, hlines}}$: draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as $\backslash \texttt{\textbackslash Cdots}$ and in the corners (if the key $\texttt{except-corners}$ is used).

The key $hvlines$

The following command tests wether the current position in the array (given by $\texttt{\backslash l\_tmpa\_tl}$ for the row and $\texttt{\backslash l\_tmpb\_tl}$ 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 $\texttt{\backslash l\_tmpa\_bool}$ is set to false.

The command $\backslash \texttt{\textbackslash Hline}$: will be linked to $\backslash \texttt{\textbackslash Hline}$ in the environments of $\texttt{nicematrix}$. The argument of the command $\backslash \texttt{\textbackslash Hline\_i:n}$ is the number of successive $\backslash \texttt{\textbackslash Hline}$ found.
\texttt{\int_compare_p:nNn \l_tmpb_tl > \{ \#2 - 1 \} }
\texttt{\int_compare_p:nNn \l_tmpb_tl < \{ \#4 + 1 \} }
\texttt{\bool_gset_false:N \g_tmpa_bool }

The same for vertical rules.

\texttt{\cs_new_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4 }
\texttt{\bool_lazy_all:nT}
\texttt{\int_compare_p:nNn \l_tmpa_tl > \{ \#1 - 1 \} }
\texttt{\int_compare_p:nNn \l_tmpa_tl < \{ \#3 + 1 \} }
\texttt{\int_compare_p:nNn \l_tmpb_tl > \{ \#2 \} }
\texttt{\int_compare_p:nNn \l_tmpb_tl < \{ \#4 + 1 \} }
\texttt{\bool_gset_false:N \g_tmpa_bool }

The key except-corners

When the key \texttt{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.

\texttt{\cs_new_protected:Npn \@@_compute_corners: }
\texttt{\seq_clear_new:N \l_@@_empty_corner_cells_seq }
\texttt{\clist_map_inline:Nn \l_@@_except_corners_clist }
\texttt{\str_case:nnF { ##1 } }
\texttt{\@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
\texttt{\@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
\texttt{\@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
\texttt{\@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
\texttt{\@@_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 \texttt{\l_@@_empty_corner_cells_seq}.

The six arguments of \texttt{\@@_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.
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 $\l_{\text{tmpa\_bool}}$ will be raised when a non-empty cell is found.

Now, you determine the last empty cell in the row of number 1.

Now, we loop over the rows.

We treat the row number 1 with another loop.
The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a \texttt{\textbackslash diagbox}).

The flag \texttt{\l_tmpb_bool} will be raised if the cell \texttt{#1}\texttt{-#2} is in a block (or in a cell with a \texttt{\textbackslash diagbox}).

\begin{verbatim}
\cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
{ \int_set:Nn \l_tmpa_int { #1 } \int_set:Nn \l_tmpb_int { #2 } \bool_set_false:N \l_tmpb_bool \seq_map_inline:Nn \g_@@_pos_of_blocks_seq { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 } }
\end{verbatim}

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 \texttt{\textbackslash col} nodes and the \texttt{\textbackslash row} nodes.

\textbf{Horizontal dotted lines}

The following command must \textit{not} be protected because it’s meant to be expanded in a \texttt{\textbackslash noalign}.

\begin{verbatim}
\cs_new:Npn \@@_hdottedline:
{ \noalign { \skip_vertical:N 2\l_@@_radius_dim } \@@_hdottedline_i: }
\end{verbatim}

On the other side, the following command should be protected.

\begin{verbatim}
\cs_new_protected:Npn \@@_hdottedline_i:
{ \tl_gput_right:Nx \g_@@_internal_code_after_tl \texttt{\@@_hdottedline:n { \int_use:N \c@iRow } } }
\end{verbatim}
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.

```latex
\AtBeginDocument
{
  \cs_new_protected:Npx \@@_hdottedline:n #1
  \begin{cs_new_protected:Npx}
    \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 \@@_hdottedline_i:n { #1 }
    \c_@@_endpgfortikzpicture_tl
  \end{cs_new_protected:Npx}
}
```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”.

```latex
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

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).

```latex
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

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 “1” (considering the rule having a width equal to the diameter of the dots).

```latex
\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`.

```latex
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```
For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by 0.5 \_@@\_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}

\subsection*{Vertical dotted lines}

\begin{verbatim}
\cs_new_protected:Npn \@@_vdottedline:n #1
{ \bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_final_open_bool
\bool_if:NTF \c_@@_tikz_loaded_bool
{ \tikzpicture
\@@_vdottedline_i:n { #1 }
\endtikzpicture
}
{ \pgfpicture
\@@_vdottedline_i:n { #1 }
\endpgfpicture
}
\end{verbatim}

We recall that, when externalization is used, \texttt{\tikzpicture} and \texttt{\endtikzpicture} (or \texttt{\pgfpicture} and \texttt{\endpgfpicture}) must be directly “visible”.

\begin{verbatim}
\bool_if:NTF \c_@@_tikz_loaded_bool
{ \tikzpicture
 \@@_vdottedline_i:n { #1 }
\endtikzpicture
}
{ \pgfpicture
 \@@_vdottedline_i:n { #1 }
\endpgfpicture
}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_vdottedline_i:n #1
{ \CT@arc@ \pgfrememberpicturepositiononpagetrue 
\@@_qpoint:n { col - \int_eval:n { #1 + 1 } }
\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 }
\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 }
\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}
The environment \{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.

As of now, there is only one option available for the environment \{NiceMatrixBlock\}.

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).

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

The extra nodes

First, two variants of the functions \dim\_min:nn and \dim\_max:nn.

135
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}`.

For each row `i`, we compute two dimensions `\l_@@_row_i_min_dim` and `\l_@@_row_i_max_dim`. The dimension `\l_@@_row_i_min_dim` is the minimal `y`-value of all the cells of the row `i`. The dimension `\l_@@_row_i_max_dim` is the maximal `y`-value of all the cells of the row `i`. Similarly, for each column `j`, we compute two dimensions `\l_@@_column_j_min_dim` and `\l_@@_column_j_max_dim`. The dimension `\l_@@_column_j_min_dim` is the minimal `x`-value of all the cells of the column `j`. The dimension `\l_@@_column_j_max_dim` 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 `\c_max_dim` or `-\c_max_dim`.

```latex
\cs_new_protected:Npn \@@_computations_for_medium_nodes:n {
  \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i: {
    \dim_zero_new:c { \l_@@_row_\@@_i: _min_dim } \c_max_dim
    \dim_zero_new:c { \l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
  }
  \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j: {
    \dim_zero_new:c { \l_@@_column_\@@_j: _min_dim } \c_max_dim
    \dim_zero_new:c { \l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
  }
}
```

We begin the two nested loops over the rows and the columns of the array.

```latex
\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: {
    \cs_if_exist:cT \pgf@sh@ns\@@_env: - \@@_i: - \@@_j: {
      \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
      \dim_set:cn { \l_@@_row_\@@_i: _min_dim } \dim_min:vn { \l_@@_row_\@@_i: _min_dim } \pgf@y
      \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: } {
        \dim_set:cn { \l_@@_column_\@@_j: _min_dim } \dim_min:vn { \l_@@_column_\@@_j: _min_dim } \pgf@x
      }
      \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
      \dim_set:cn { \l_@@_row_\@@_i: _max_dim } \dim_max:vn { \l_@@_row_\@@_i: _max_dim } \pgf@y
    }
  }
}
```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell `(i-j)`. They will be stored in `\pgf@x` and `\pgf@y`.

```latex
\pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
\dim_set:cn { \l_@@_row_\@@_i: _min_dim } \dim_min:vn { \l_@@_row_\@@_i: _min_dim } \pgf@y
\seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: } {
  \dim_set:cn { \l_@@_column_\@@_j: _min_dim } \dim_min:vn { \l_@@_column_\@@_j: _min_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 `\pgf@x` and `\pgf@y`.

```latex
\pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
\dim_set:cn { \l_@@_row_\@@_i: _max_dim } \dim_max:vn { \l_@@_row_\@@_i: _max_dim } \pgf@y
```

136
Now, we have to deal with empty rows or empty columns since we don’t have created nodes in such rows and columns.

\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
{\dim_compare:nNnT}
{\dim_use:c { l_@@_row _ \@@_i: _ min _ dim }} = \c_max_dim

{\@@_qpoint:n { row - \@@_i: - base } \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y}

\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
{\dim_compare:nNnT}
{\dim_use:c { l_@@_column _ \@@_j: _ min _ dim }} = \c_max_dim

{\@@_qpoint:n { col - \@@_j: } \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y}

Here is the command \@@_create_medium_nodes: . When this command is used, the “medium nodes” are created.

\cs_new_protected:Npn \@@_create_medium_nodes:
{\pgfpicture \pgfrememberpicturepositiononpagetrue \pgf@relevantforpicturesizefalse \@@_computations_for_medium_nodes:

\l_set:Nn \l_@@_suffix_tl { -medium }
\@@_create_nodes:
\endpgfpicture}

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:
\endpgfpicture}

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:.

\cs_new_protected:Npn \@@_create_large_nodes:
{\pgfpicture \pgfrememberpicturepositiononpagetrue \pgf@relevantforpicturesizefalse \@@_computations_for_medium_nodes:
\@@_computations_for_large_nodes:
\endpgfpicture}

43If we want to create both, we have to use \@@_create_medium_and_large_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_medium_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:
{\int_set:Nn \l_@@_first_row_int 1
\int_set:Nn \l_@@_first_col_int 1

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:
{\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: _ min _ dim }
\dim_set_eq:cc { l_@@_row_\@@_i: _ max _ dim }

\int_step_variable:nNn { \c@jCol - 1 } \@@_j:
{\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 }
\dim_set_eq:cc { l_@@_column_\@@_j: _ max _ dim }

Here, we have to use \dim_sub:cn because of the number 1 in the name.
\dim_sub:cn
{ l_@@_column_ \c@jCol - 1 _ min _ dim }
\l_@@_left_margin_dim
\dim_add:cn
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 \texttt{\multicolumn}. We recall that we have stored in \texttt{\g_@@_multicolumn_cells_seq} the list of the cells where a \texttt{\multicolumn{\ldots}{\ldots}{\ldots}} with \texttt{n}>1 was issued and in \texttt{\g_@@_multicolumn_sizes_seq} the correspondent values of \texttt{n}.

The command \@@_node_for_multicolumn:nn takes two arguments. The first is the position of the cell where the command \texttt{\multicolumn(n){\ldots}{\ldots}} was issued in the format \texttt{i-j} and the second is the value of \texttt{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 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 (probably because of the first optional argument?).

\NewExpandableDocumentCommand \@@_Block: { O { } m D < > { } m } \@@_Block_i #2 \q_stop { #1 } { #3 } { #4 } \@@_Block_ii:nnnnn #1 #2 #3 #4 #5

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.

\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } { #3 } { #4 } { #5 } }

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.

\cs_new:vpn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

Now, \l_tmpa_tl contains an “object” corresponding to the position of the block whith four components surrounded by curly brackets:

\{imin\}{jmin\}{imax\}{jmax\}

We store this information in the sequence \l_@@_pos_of_blocks_seq.

We also store a complete description of the block in the sequence \l_@@_blocks_seq. Of course, the sequences \l_@@_pos_of_blocks_seq and \l_@@_blocks_seq are redundant, but it’s for efficiency.

In \l_@@_blocks_seq, each block is represented by an “object” with six components:

\{imin\}{jmin\}{imax\}{options\}{contents\}

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?

\keys_define:nn { NiceMatrix / Block }
The command `\@@_draw_blocks` will draw all the blocks. This command is used after the construction of the array.

```
\cs_new_protected:Npn \@@_draw_blocks: { \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iii:nnnnnn ##1 } }
```

The group is for the keys.

```
\group_begin:
\keys_set:nn { NiceMatrix / Block } { #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 }
}
```

```
\bool_lazy_or:nnTF
{ \int_compare_p:nNn { #3 } > \g_@@_row_total_int }
{ \int_compare_p:nNn { #4 } > \g_@@_col_total_int }
{ \msg_error:nnnn { nicematrix } { Block-too-large } { #1 } { #2 } }
```

We put the contents of the cell in the box `\l_@@_cell_box` because we want the command `\rotate` used in the content to be able to rotate the box.

```
\hbox_set:Nn \l_@@_cell_box { #6 }
```

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

\begin{tabular}{|c|c|}
\hline
our block & one \\
\hline
three & four \\
two & five \\
six & seven \\
seven & eight \\
\hline
\end{tabular}

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}
\dim_set_eq:NN \l_tmpb_dim \c_max_dim
\int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int 
{ }
\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.

\begin{cs_if_exist:cT}
{ \pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
{ \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 } { west } 
\dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
}
\end{cs_if_exist:cT}

If all the cells of the column were empty, \l_tmpb_dim has still the same value \c_max_dim. In that case, you use for \l_tmpb_dim the value of the position of the vertical rule.

\begin{cs_if_exist:cT}
{ \pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
{ \dim_set_eq:NN \l_tmpb_dim \pgf@x }
\end{cs_if_exist:cT}

\begin{cs_if_exist:cT}
{ \pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
{ \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
}
\end{cs_if_exist:cT}
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
\@@_pgf_rect_node:nnn
{ \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
{ \pgfpointanchor { \@@_env: - #3 - #4 - medium } { south-east } }

Now, we will put the label of the block beginning with the case of a Block of one row.
\int_compare:nNnTF { #1 } = { #3 }
{ \l_@@_code_for_first_row_tl }
{ \int_compare:nNnT { #1 } = \l_@@_last_row_int
\l_@@_code_for_last_row_tl }

We take into account the case of a block of one row in the "first row" or the "end row".
\int_compare:nNnTF { #1 } = 0
{ \l_@@_code_for_first_row_tl }
{ \int_compare:nNnT { #1 } = \l_@@_last_row_int
\l_@@_code_for_last_row_tl }

If the block has only one row, we want the label of the block perfectly aligned on the baseline of the row. That's why we have constructed a \pgfcoordinate on the baseline of the row, in the first column of the array. Now, we retrieve the y-value of that node and we store it in \l_tmpa_dim.
\pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

We retrieve (in \pgf@x) the x-value of the center of the block.
\pgfpointanchor
{ \@@_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 }
If the number of rows is different of 1, we put the label of the block in using the short node (the label of the block has been composed in \l@_cell_box).

{ \pgftransformshift
  { \pgfpointanchor{\@@_env: - #1 - #2 - block - short}{\str_case:Vn\l@_pos_of_block_tl}
    {\str_case:Vn\l@_pos_of_block_tl}
      {\str_case:Vn\l@_pos_of_block_tl}
        {\box_use_drop:N\l@_cell_box}{}{}
    }
}
{ \pgfset{inner sep = \c_zero_dim}
  \pgfnode{rectangle}{\str_case:Vn\l@_pos_of_block_tl}
    {\str_case:Vn\l@_pos_of_block_tl}
      {\str_case:Vn\l@_pos_of_block_tl}
        {\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}{}\endpmatrix}
  {\RenewDocumentEnvironment{vmatrix}{}\endvmatrix}
  {\RenewDocumentEnvironment{Vmatrix}{}\endVmatrix}
  {\RenewDocumentEnvironment{bmatrix}{}\endbmatrix}
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
{ \{ \} \#6 & } #6
\int_compare:nNnT \l_@@_last_row_int > { -2 }
{ \int_compare:nNnT \l_@@_first_col_int = 0 { & }
\prg_replicate:nn \l_@@_nb_cols_int
{ \{ \} \#6 & } #6
\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 }
\str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
\AutoNiceMatrixWithDelims { #2 } { #3 }
\@@_define_com:nnn p ( )
\@@_define_com:nnn b [ ]
\@@_define_com:nnn v | |
\@@_define_com:nnn V \| |
\@@_define_com:nnn B \{ \}

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

145
The redefinition of the command \dotfill

\cs_set_eq:NN \@@_old_dotfill \dotfill
\cs_new_protected:Npn \@@_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}).
\@@_old_dotfill
\bool_if:NT \l_@@_NiceTabular_bool
{ \group_insert_after:N \@@_dotfill_ii: }
{ \group_insert_after:N \@@_dotfill_i: }
}
\cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
\cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

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.
\seq_gput_right:Nx \g_@@_pos_of_blocks_seq
{ { \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
}

The command \diagbox is also redefined locally when we draw a block.

The first four arguments of \@@_actually_diagbox:nnnnnn correspond to the rectangle (=block) to slash (we recall that it’s possible to use \diagbox in a \Block). The two other are the elements to draw below and above the diagonal line.
\cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
{ \pgfpicture
\pgf@relevantforpicturesizefalse
\pgfresetpicturepositiononpagetrue
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.

\CT@arc@
\pgfsetroundcap
\pgfusepathqstroke
\pgfset { inner~sep = 1 pt }
\pgfscope
\pgfnode { rectangle } { south~west }
{ \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
\endpgfscope
\pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
\pgfnode { rectangle } { north~east }
{ \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
\endpgfpicture

The keyword \CodeAfter

In fact, in this subsection, we define the user command \CodeAfter for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment \@@-light-syntax on p. 85.

The command \CodeAfter catches everything until the end of the current environment (of nicematrix). First, we go until the next command \end.

\cs_new_protected:Npn \@@_CodeAfter:n #1 \end
{ \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 } \@@_CodeAfter_i:n }

We catch the argument of the command \end (in #1).

\cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
{ \str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } } \@@_CodeAfter:n }

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.

\str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } } \@@_CodeAfter:n
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.

{ \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } } \@@_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.

\bool_new:N \c_@@_footnotehyper_bool

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.

\bool_new:N \c_@@_footnote_bool

4515 \@@_msg_new:nnn { Unknown~option~for~package }
4516 \{ 4517 The~option~'l_keys_key_tl'~is~unknown. \ 4518 If~you~go~on,~it~will~be~ignored. \ 4519 For~a~list~of~the~available~options,~type~H~<return>. \ 4520 } 4521 \keys_define:nnn { NiceMatrix / Package } 4522 { 4523 define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool , 4524 define-L-C-R .default:n = true , 4525 renew-dots .bool_set:N = \l_@@_renew_dots_bool , 4526 renew-dots .value_forbidden:n = true , 4527 renew-matrix .code:n = \@@_renew_matrix: , 4528 renew-matrix .value_forbidden:n = true , 4529 transparent .meta:n = { renew-dots , renew-matrix } , 4530 transparent .value_forbidden:n = true , 4531 footnote .bool_set:N = \c_@@_footnote_bool , 4532 footnotehyper .bool_set:N = \c_@@_footnotehyper_bool , 4533 unknown .code:n = \@@_error:n { Unknown~option~for~package } 4534 } 4535 \ProcessKeysOptions { NiceMatrix / Package }

4546 \@@_msg_new:nnn { footnote~with~footnotehyper~package }
4547 \{ 4548 You~can't~use~the~option~'footnote'~because~the~package~footnotehyper~has~already~been~loaded.~ 4549 If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~of~the~package~footnotehyper.\ 4550 If~you~go~on,~the~package~footnote~won't~be~loaded. \ 4551 } 4552 \@@_msg_new:nnn { footnotehyper~with~footnote~package }
4553 \{ 4554 You~can't~use~the~option~'footnotehyper'~because~the~package~footnote~has~already~been~loaded.~ 4555 If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~of~the~package~footnote.\}
If you go on, the package footnotehyper won’t be loaded.

\bool_if:NT \c_@@_footnote_bool
\{
\@ifclassloaded { beamer }
\{ \msg_info:nn { nicematrix } { Option-incompatible-with-Beamer } \}
\{ \@ifpackageloaded { footnotehyper }
\{ \@@_error:n { footnote-with-footnotehyper-package } \}
\{ \usepackage { footnote } \}
\}
\bool_if:NT \c_@@_footnotehyper_bool
\{
\@ifclassloaded { beamer }
\{ \@@_info:n { Option-incompatible-with-Beamer } \}
\{ \@ifpackageloaded { footnote } \@@_error:n { footnotehyper-with-footnotehyper-package } \}
\{ \usepackage { footnotehyper } \}
\}
\bool_set_true:N \c_@@_footnote_bool
\}
\}

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
\{ \seq_put_left:Nx \l_tmpa_seq { \tl_to_str:n { ##1 } } \}
\seq_set_eq:NN #1 \l_tmpa_seq
\}

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 \g_@@_name_env_str
\}
\@@_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:NVTF 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
\}

149
The following command must not be protected since it’s used in an error message.

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.

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.
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { 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. \}
\@_msg_new:nn { double-backslash-in-light-syntax }
\{ You can't use \token_to_str:N \ to separate rows because you have used the option 'light-syntax'. You must use the character '\l_@@_end_of_row_tl' (set by the option 'end-of-row'). This error is fatal. \}
\@_msg_new:nn { standard-cline-in-document }
\{ The key 'standard-cline' is available only in the preamble.\\
If you go on this command will be ignored. \}
\@_msg_new:nn { bad-value-for-baseline }
\{ The value given to 'baseline' (\int_use:N \l_tmpa_int) is not valid. The value must be between \int_use:N \l_@@_first_row_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. \}
\@_msg_new:nn { empty-environment }
\{ Your \@_full_name_env:\ is empty. This error is fatal. \}
\@_msg_new:nn { unknown-cell-for-line-in-code-after }
\{
Your command in the 'code-after' can't be executed because a cell doesn't exist. If you go on, this command will be ignored.

\@@_msg_new:nn {Hdotsfor-in-col-0}
\{ You can't use \Hdotsfor\ in an exterior column of the array. If you go on, the corresponding dotted line won't be drawn. \}

\@@_msg_new:nn {bad-corner}
\{ #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. \}

\@@_msg_new:nn {last-col-non-empty-for-NiceArray}
\{ 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). \}

\@@_msg_new:nn {last-col-non-empty-for-NiceMatrixOptions}
\{ 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). \}

\@@_msg_new:nn {Block-too-large}
\{ You try to draw a block in the cell #1-#2 of your matrix but the matrix is too small for that block. \}

\@@_msg_new:nn {unknown-column-type}
\{ The column type '#1' in your \@@_full_name_env: is unknown. This error is fatal. \}

\@@_msg_new:nn {angle-option-in-NiceTabular}
\{ You should not use the option between angle brackets (< and >) for a command \token_to_str:N\Block in \{NiceTabular\}. However, you can go on. \}

\@@_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 {bottomrule-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-\token_to_str:N \Block.\\
  If-you-go-on,-it-will-be-ignored. \\
  For-a-list-of-the-available-keys,-type-H<return>.\}
{
  The-available-options-are-(in-alphabetic-order):-c,-
  color,-l,-and-r.
}
\_\_\_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>.\}
{
  The-available-options-are-(in-alphabetic-order):-
  bottomrule,-
  code-after,-
  code-before,-
  enumitem-keys,-
  enumitem-keys-para,-
  para,-
  label-in-list,-
  label-in-tabular-and-
  style.
}
\_\_\_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>.\}
The available options are (in alphabetic order):

allow-duplicate-names,-
cell-space-bottom-limit,-
cell-space-top-limit,-
code-for-first-col,-
code-for-first-row,-
code-for-last-col,-
code-for-last-row,-
create-extra-nodes,-
create-medium-nodes,-
create-large-nodes,-
end-of-row,-
first-col,-
first-row,-

hlines,-
hlines,-
hlines-except-corners,-
last-col,-
last-row,-
left-margin,-
letter-for-dotted-lines,-
light-syntax,-
notes-(several subkeys),-
nullify-dots,-
renew-dots,-
renew-matrix,-
right-margin,-
small,-
transparent,-
vlines,-
xdots/color,-
xdots/shorten-and-
xdots/line-style.

\@@_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\>.

{ 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,-
colortbl-like,-
columns-width,-
create-extra-nodes,-
create-medium-nodes,-
create-large-nodes,-
extra-left-margin,-
extra-right-margin,-
first-col,-
This error message is used for the set of keys \texttt{NiceMatrix/NiceMatrix} and \texttt{NiceMatrix/pNiceArray} (but not by \texttt{NiceMatrix/NiceArray} because, for this set of keys, there is also the options \texttt{t, c and b}).

\begin{verbatim}
\@@_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):
\texttt{b,}-
\texttt{baseline,}-
\texttt{c,}-
\texttt{cell-space-bottom-limit,}-
\texttt{cell-space-top-limit,}-
\texttt{code-after,}-
\texttt{code-for-first-col,}-
\texttt{code-for-first-row,}-
\texttt{code-for-last-col,}-
\texttt{code-for-last-row,}-
\texttt{colortbl-like,}-
\texttt{columns-width,}-
\texttt{create-extra-nodes,}-
\texttt{create-medium-nodes,}-
\texttt{create-large-nodes,}-
\texttt{extra-left-margin,}-
\texttt{extra-right-margin,}-
\texttt{first-col,}-
\texttt{first-row,}-
\texttt{hlines,}-
\texttt{l,}-
\texttt{last-col,}-
\texttt{last-row,}-
\texttt{left-margin,}-
\texttt{light-syntax,}-
\texttt{name,}-
\texttt{nullify-dots,}-
\texttt{r,}-%
\end{verbatim}

155
\@@_msg_new:nnn { Unknown-option-for-NiceTabular }
{\begin{verbatim}
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>.
\end{verbatim}
}

\@@_msg_new:nnn { Duplicate-name }
{\begin{verbatim}
The-name-’\l_keys_value_tl’-is-already-used-and-you-shouldn’t-use-
the-same-environment-name-twice.-You-can-go-on,-but,-
\end{verbatim}
}
maybe, you will have incorrect results especially if you use 'columns-width=auto'. If you don't want to see this message again, use the option 'allow-duplicate-names'in 
\token_to_str:N \NiceMatrixOptions'.

For a list of the names already used, type H<return>.
}\)
{ \The names already defined in this document are: -
\seq_use:Nnnn \g_@@_names_seq { , } { , } { and }.
}\}
\@@_msg_new:nn { Option-auto-for-columns-width }
{ You can't give the value 'auto' to the option 'columns-width' here.
If you go on, the option will be ignored.
}\}

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 \{NiceArray\} with column types L, C and R.

Changes between version 1.2 and 1.3

New environment \{pNiceArrayC\} and its variants.
Correction of a bug in the definition of \{BNiceMatrix\}, \{vNiceMatrix\} and \{vNiceMatrix\} (in fact, it was a typo).
Options are now available locally in \{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 \{NiceArray\}, \{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 nicematrix were focused on the continuous dotted lines whereas the version 2.0 of 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{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{\NiceMatrixOptions}.
Following a discussion on TeX StackExchange\textsuperscript{44}, 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{\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{\Cdots} command (and a single \texttt{\Vdots}).

\[
\begin{pmatrix}
C_j \\
0 & \cdots & 0 \\
0 & a & \cdots & 0
\end{pmatrix}
L_i
\]

Changes between version 2.1.3 and 2.1.4

Replacement of some options \texttt{O \{ \}} in commands and environments defined with \texttt{xparse} by \texttt{! O \{ \}}
(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-optional-arguments-at-the-end}

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes \texttt{revtex4-1} and \texttt{revtex4-2}.
Option \texttt{allow-duplicate-names}.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command \texttt{\hdottedline} (similar to the classical command \texttt{\hline} and the command \texttt{\dashline} of \texttt{arydshln}).
Possibility to draw vertical dotted lines to separate columns with the specifier “\texttt{:}” in the preamble
(similar to the classical specifier “\texttt{\mid}” and the specifier “\texttt{:}” of \texttt{arydshln}).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “\texttt{:}” in the preamble.
Modification of the position of the dotted lines drawn by \texttt{\hdottedline}.

\textsuperscript{44}\texttt{cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}

\textsuperscript{45}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 2.2.1 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 \cdottedline, the symbol “;” (in the preamble of the array) and \line in code-after).
The starred versions of \Cdots, \Ldots, 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{46}, 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.

\textsuperscript{46}cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize
Changes between version 3.5 and 3.6

LaTeX counters \row and \col available in the cells of the array.
Addition of \normalbaselines before the construction of the array: in environments like {align} of amsmath the value of \baselineskip is changed and if the options first-row and last-row were used in an environment of nicematrix, the position of the delimiters was wrong.
A warning is written in the .log file if an obsolete environment is used.
There is no longer artificial errors Duplicate-name in the environments of amsmath.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: code-for-first-col, code-for-last-col, code-for-first-row and code-for-last-row.
New command \pAutoNiceMatrix and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command \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 \multicolumn.
An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands \NiceMatrixLastEnv and \OnlyMainNiceMatrix.
New options create-medium-nodes and create-large-nodes.

Changes between version 3.9 and 3.10

New option light-syntax (and end-of-row).
New option dotted-lines-margin for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to first-row and last-row.

Changes between versions 3.11 and 3.12

Command \rotate in the cells of the array.
Options vlines, hlines and hvlines.
Option baseline pour {NiceArray} (not for the other environments).
The name of the Tikz nodes created by the command \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 try to use more columns than allowed by its environment, an error is raised by nicematrix (instead of a low-level error).
The package must be loaded with the option 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 dotted-lines-margin has been renamed in xdots/shorten and the options xdots/color and xdots/line-style have been added for a complete customization of the dotted lines.

In the environments without preamble ({NiceMatrix}, {pNiceMatrix}, etc.), it’s possible to use the options 1 (=L) or r (=R) to specify the type of the columns.

The starred versions of the commands \Cdots, \Ldots, \Vdots, \Ddots and \Iddots are deprecated since the version 3.1 of nicematrix. Now, one should load nicematrix with the option starred-commands to avoid an error at the compilation.

The code of nicematrix no longer uses Tikz but only PGF. By default, Tikz is not loaded by nicematrix.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on stackoverflow).

Better error messages when the user uses & or \ when 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 \Ldots, \Cdots, \Vdots, \Ddots, \Iddots, \Hdots for and the command \line in the code-after with the tokens _ and ^.

The option baseline is now available in all the environments of nicematrix. Before, it was available only in {NiceArray}.

New keyword \CodeAfter (in the environments of nicematrix).

Changes between versions 3.15 and 4.0

New environment {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 cell-space-top-limit and cell-space-bottom-limit

New command \diagbox

The key hvline don’t draw rules in the blocks (commands \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 \begin{pNiceMatrix}a&b\c&d\end{pNiceMatrix}^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 \qquad 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 hvlines-except-corners.
Changes between versions 4.4 and 5.0

Use of the standard column types l, c and r instead of L, C and R.
It's now possible to use the command \diagbox in a \Block.
Command \tabularnote

Changes between versions 5.0 and 5.1

The vertical rules specified by | in the preamble are not broken by \hline\hline (and other).
Environment \{NiceTabular*}\nCommand \Vdotsfor similar to \Hdotsfor
The variable \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 respect-blocks for rowcolors (with a s) in the code-before.
The variable \g_nicematrix_code_before_tl is now public.
The key baseline can take in as value of the form line-i to align the \hline in the row i.
The key hlines-except-corners may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys c, r and 1 for the command \Block.
It's possible to use the key draw-first with \Ddots and \Iddots to specify which dotted line will
be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

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></th>
</tr>
</thead>
<tbody>
<tr>
<td>\g_@@_Hspace:</td>
<td>1004, 2925</td>
</tr>
<tr>
<td>\g_@@_Hline_iii:n</td>
<td></td>
</tr>
<tr>
<td>\g_@@_Hline_ii:nn</td>
<td></td>
</tr>
<tr>
<td>\g_@@_Hline_i:n</td>
<td></td>
</tr>
<tr>
<td>\g_@@_Hdotsfor_lines_tl</td>
<td>1029, 2167, 2992, 3068, 4620</td>
</tr>
<tr>
<td>\g_@@_Hdotsfor:</td>
<td>938, 1005, 2977</td>
</tr>
<tr>
<td>\g_@@_Hdotsfor:mmnn</td>
<td>2994, 3006</td>
</tr>
<tr>
<td>\g_@@_Hdotsfor:i</td>
<td>2983, 2990</td>
</tr>
<tr>
<td>\g_@@_Hline:</td>
<td>1003, 3727</td>
</tr>
<tr>
<td>\g_@@_Hline_i:n</td>
<td>3727, 3728, 3734</td>
</tr>
<tr>
<td>\g_@@_Hline_ii:nn</td>
<td>3731, 3734</td>
</tr>
<tr>
<td>\g_@@_Hline_iii:nn</td>
<td>3732, 3735</td>
</tr>
</tbody>
</table>

\|\g_@@_Iddots | 936, 1001, 2895 |
\|\g_@@_Iddots_lines_tl | 1028, 2170 |
\|\g_@@_Ldots | 932, 937, 997, 2827 |
\|\g_@@_Ldots_lines_tl | 1025, 2172 |
\|\l_@@_Matrix_bool | 237, 1228, 1302, 1962 |
\|\l_@@_NiceArray_bool | |
\|\g_@@_NiceMatrixBlock_int | 234, 323, 1069, 1180, 1240, 1333, 1345, 1938, 3066, 3070, 3723, 3724, 4434 |
\|\g_@@_NiceMatrixBlock_int | 230, 3965, 3970, 3973, 3984 |
\|\l_@@_NiceTabular_bool | |
\|\l_@@_NiceMatrix_i:n | 3472, 3479, 3482 |
\|\l_@@_NiceMatrix_i:n | 3472, 3479, 3482 |
\|\l_@@_NiceMatrix_i:n | 934, 999, 2857 |
\|\g_@@_Vdotsfor_lines_tl | 1026, 2168 |
\|\g_@@_Vdotsfor: | 1006, 3066 |
\|\g_@@_Vdotsfor:mmnn | 3070, 3081 |
\|\g_@@_Vdotsfor:i | 1306, 1375 |
\|\g_@@_Vdotsfor:mmnn | 4230, 4454, 4470 |
\|\g_@@_Vdotsfor:mmnn | 2418, 2422 |
\|\g_@@_Vdotsfor:mmnn | 2546, 2550 |
\|\g_@@_Vdotsfor:mmnn | 2597, 2601 |
The environments of this package

The vertical space between the rows

The vertical position of the arrays

The blocks

The rules

5.1 Some differences with the classical environments

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

5  The rules ........................................................................... 5

172
5.1.1 The vertical rules ........................................... 5
5.1.2 The command \cline ........................................ 6
5.2 The thickness and the color of the rules ......................... 6
5.3 The keys hlines and vlines ...................................... 6
5.4 The key hvlines ................................................. 7
5.5 The key hvlines-except-corners ................................ 7
5.6 The command \diagbox ........................................... 8
5.7 Dotted rules .................................................... 8

6 The color of the rows and columns ................................. 9
6.1 Use of colorbl .................................................. 9
6.2 The tools of nicematrix in the code-before ..................... 9
6.3 Color tools with the syntax of colorbl .......................... 11

7 The width of the columns ........................................ 13

8 The exterior rows and columns .................................... 14

9 The continuous dotted lines ....................................... 15
9.1 The option nullify-dots ....................................... 17
9.2 The commands \Hdotsfor and \Vdotsfor .......................... 17
9.3 How to generate the continuous dotted lines transparently ... 18
9.4 The labels of the dotted lines ................................ 19
9.5 Customization of the dotted lines .............................. 19
9.6 The dotted lines and the rules ................................ 20

10 The code-after .................................................. 20

11 The notes in the tabulars ......................................... 21
11.1 The footnotes .................................................. 21
11.2 The notes of tabular ......................................... 21
11.3 Customisation of the tabular notes ............................. 23
11.4 Use of \{NiceTabular\} with threeparttable ................... 24

12 Other features .................................................. 25
12.1 Use of the column type S of siunitx ........................... 25
12.2 Alignment option in \{NiceMatrix\} ............................. 25
12.3 The command \rotate .......................................... 25
12.4 The option small .............................................. 26
12.5 The counters iRow and jCol .................................. 26
12.6 The option light-syntax ...................................... 27
12.7 The environment \{NiceArrayWithDelims\} ....................... 28

13 Use of Tikz with nicematrix .................................... 28
13.1 The nodes corresponding to the contents of the cells ......... 28
13.2 The “medium nodes” and the “large nodes” .................... 29
13.3 The “row-nodes” and the “col-nodes” ........................ 30

14 API for the developers .......................................... 31

15 Technical remarks ............................................... 32
15.1 Definition of new column types ............................... 32
15.2 Diagonal lines ................................................ 32
15.3 The “empty” cells .............................................. 33
15.4 The option exterior-arraycolsep ................................ 33
15.5 Incompatibilities ............................................. 33
<table>
<thead>
<tr>
<th>16</th>
<th>Examples</th>
<th>34</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

| 17   | Implementation                | 42 |

| 18   | History                       | 157 |

Index 162