The package \texttt{nicematrix}\textsuperscript{*}

F. Pantigny
fpantigny@wanadoo.fr
March 23, 2021

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}{c c c c}
\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 \\
\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 TeXLive.

Remark: If you use LaTeX via Internet with, for example, Overleaf, you can upload the file \texttt{nicematrix.sty} in the repertory of your project in order to take full advantage of the latest version of \texttt{nicematrix}.

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). \textit{However, the file nicematrix.dtx of the present documentation should be compiled with XeLaTeX.}

This package requires and \texttt{loads} the packages \texttt{3keys2e}, \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 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 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 several compilations.

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

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

\textsuperscript{*}This document corresponds to the version 5.13 of \texttt{nicematrix}, at the date of 2021/03/23.

\textsuperscript{1}The latest version of the file \texttt{nicematrix.sty} may be downloaded from the \texttt{svn} server of TeXLive: \url{https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty}
The environments of this package

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

\begin{itemize}
\item \texttt{NiceTabular}
\item \texttt{NiceArray}
\item \texttt{NiceMatrix}
\item \texttt{NiceTabular*}
\item \texttt{pNiceArray}
\item \texttt{pNiceMatrix}
\item \texttt{bNiceArray}
\item \texttt{bNiceMatrix}
\item \texttt{BNiceArray}
\item \texttt{BNiceMatrix}
\item \texttt{vNiceArray}
\item \texttt{vNiceMatrix}
\item \texttt{VNiceArray}
\item \texttt{VNiceMatrix}
\end{itemize}

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

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

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

\section{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}
\end{verbatim}

\begin{verbatim}
\begin{pNiceMatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14 \\
\end{pNiceMatrix}
\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}.

There is also a key \texttt{cell-space-limits} to set both parameters at once. 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-limits = 1pt}}.

\begin{verbatim}
\NiceMatrixOptions{cell-space-limits = 1pt}
\begin{pNiceMatrix}
\frac12 & -\frac12 \\
\frac13 & \frac14 \\
\end{pNiceMatrix}
\end{verbatim}

\textsuperscript{2}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 \texttt{nicematrix} provides an option \texttt{baseline} for the vertical position of the arrays. This option takes as value an integer which is the number of the row on which the array will be aligned.

\begin{align*}
& \text{\texttt{\begin{NiceMatrix}[baseline=2]}} \\
& \text{\texttt{\frac{1}{\sqrt{1+p^2}} \ & p \ & 1-p \ \}} \\
& \text{\texttt{1 \ & 1 \ & 1 \ \}} \\
& \text{\texttt{1 \ & p \ & 1+p \ \}} \\
& \text{\texttt{\end{NiceMatrix}$}}
\end{align*}

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 \{\texttt{tabular}\} and \{\texttt{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 \{\texttt{tabular}\} or \{\texttt{array}\} of \texttt{array}, one must use \texttt{\firsthline}).

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

It’s also possible to use the key \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.

\begin{NiceMatrixOptions{cell-space-limits=1pt}
\$\begin{pNiceArray}{cc|cc}[baseline=\texttt{line-3}]\]
\dfrac{1}{A} & \dfrac{1}{B} & 0 & 0 \\
A & B & 0 & 0 \\
1 & C & D & 0 \\
0 & 0 & A & B \\
0 & 0 & D & D \\
\end{pNiceArray}$
\end{NiceMatrixOptions{cell-space-limits=1pt}
4 The blocks

4.1 General case

In the environments of nicematrix, it’s possible to use the command \Block in order to place an element in the center of a rectangle of merged cells of the array. The command \Block must be used in the upper leftmost cell of the array with two arguments.

- The first argument is the size of the block with the syntax $i-j$ where $i$ is the number of rows of the block and $j$ its number of columns.
  If this argument is empty, its default value is 1-1. If the number of rows is not specified, or equal to *, the block extends until the last row (idem for the columns).
- The second argument is the content of the block. It’s possible to use \ in that content to have a content on several lines. In \NiceTabular{} the content of the block is composed in text mode whereas, in the other environments, it is composed in math mode.

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

\begin{bNiceArray}{ccc|c}
\Block{3-3}{A} & & & 0 \\
& & & \Vdots \\
& & & 0 \\
\hline 
0 & \Cdots & 0 & 0
\end{bNiceArray}

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

\begin{bNiceArray}{ccc|c}
\Block{3-3}<\Large>{A} & & & 0 \\
& & & \Vdots \\
& & & 0 \\
\hline 
0 & \Cdots & 0 & 0
\end{bNiceArray}

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

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

In fact, the command \Block accepts as first optional argument (between square brackets) a list of couples key-value. The available keys are as follows:

- the keys l, c and r are used to fix the horizontal position of the content of the block, as explained previously;
- the key fill takes in as value a color and fills the block with that color;

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

\footnote{This argument between angular brackets may also be used to insert a command of font such as \bfseries when the command \ is used in the content of the block.}
the key \texttt{draw} takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);\par
the key \texttt{color} takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;\par
the key \texttt{line-width} is the width (thickness) of the frame (this key should be used only when the key \texttt{draw} is in force);\par
\textbf{New 5.12} the key \texttt{rounded-corners} requires rounded corners (for the frame drawn by \texttt{draw} and the shape drawn by \texttt{fill}) with a radius equal to the value of that key (the default value is 4 pt\textsuperscript{4}).\par
\textbf{New 5.13} the key \texttt{borders} provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by \texttt{left}, \texttt{right}, \texttt{top} and \texttt{bottom}.

One must remark that, by default, the commands \texttt{Blocks don't create space}. There is exception only for the blocks mono-row and the blocks mono-column as explained just below.

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

\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose & tulipe & marguerite & dahlia \\
violette & \Block[draw=red,fill=red!15,rounded-corners]{2-2}{\LARGE De très jolies fleurs} & & souci \\
pervenche & & lys \\
arum & iris & jacinthe & muguet
\end{NiceTabular}

\begin{tabular}{|c|c|c|c|}
\hline
rose & tulipe & marguerite & dahlia \\
violette & De très jolies fleurs & souci \\
pervenche & & lys \\
arum & iris & jacinthe & muguet \\
\hline
\end{tabular}

\subsection*{4.2 The mono-column blocks}

The mono-column blocks have a special behaviour.

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

\textsuperscript{4}This value is the initial value of the \texttt{rounded corners} of Tikz.
\begin{NiceTabular}{@{}lr@{}} \hline
John & 12 \\ & 13 \\ Steph & 8 \\ \hline
Steph & 8 \\ & 13 \\ \hline
Sarah & 18 \\ & 17 \\ Ashley & 20 \\ Henry & 14 \\ Madison & 15 \\ \hline
Madison & 15 \\ & 19 \\
\end{NiceTabular}

4.3 The mono-row blocks

For the mono-row blocks, the natural height and depth are taken into account for the height and depth of the current row (as does a standard \texttt{\multicolumn} of \LaTeX{}).

4.4 The mono-cell blocks

A mono-cell block inherits all the properties of the mono-row blocks and mono-column blocks.

At first sight, one may think that there is no point using a mono-cell block. However, there are some good reasons to use such a block.

- It’s possible to use the command \texttt{\backslash{}\backslash{}} in a (mono-cell) block.
- It’s possible to use the option of horizontal alignment of the block in derogation of the type of column given in the preamble of the array.
- It’s possible do draw a frame around the cell with the key \texttt{\texttt{draw}} of the command \texttt{\Block} and to fill the background with rounded corners with the keys \texttt{fill} and \texttt{rounded-corners}.

\begin{NiceTabular}{cc}
\toprule
Writer & \texttt{\Block}[l]{}{year\ of\ birth} \\
\midrule
Hugo & 1802 \\
Balzac & 1799 \\
\bottomrule
\end{NiceTabular}

We recall that if the first mandatory argument of \texttt{\Block} is left blank, the block is mono-cell.

4.5 A small remark

One should remark that the horizontal centering of the contents of the blocks is correct even when an instruction such as \texttt{!\textbackslash{}\textbackslash{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 \texttt{\multicolumn}). In the following example, the header “First group” is correctly centered.

\begin{NiceTabular}{cc}
\toprule
Writer & year of birth \\
\midrule
Hugo & 1802 \\
Balzac & 1799 \\
\bottomrule
\end{NiceTabular}

\footnote{If one simply wishes to color the background of a unique cell, there is no point using the command \texttt{\Block}: it’s possible to use the command \texttt{\cellcolor} (when the key \texttt{colorbl-like} is used).}

\footnote{One may consider that the default value of the first mandatory argument of \texttt{\Block} is 1-1.}
\begin{NiceTabular}{@{}c!{\quad}ccc!{\quad}ccc@{}}
\toprule
& \Block{1-3}{First group} & & & \Block{1-3}{Second group} \\
\midrule
1 & 0.657 & 0.913 & 0.733 & 0.830 & 0.387 & 0.893 \\
2 & 0.343 & 0.537 & 0.655 & 0.690 & 0.471 & 0.333 \\
3 & 0.783 & 0.885 & 0.015 & 0.306 & 0.643 & 0.263 \\
4 & 0.161 & 0.708 & 0.386 & 0.257 & 0.074 & 0.336 \\
\bottomrule
\end{NiceTabular}

5 The rules

The usual techniques for the rules may be used in the environments of \texttt{ninematrix} (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{ninematrix}, the vertical rules specified by \texttt{|} in the preambles of the environments are never broken, even by an incomplete row or by a double horizontal rule specified by \texttt{\hline}\texttt{\hline} (there is no need to use \texttt{\hhline}).

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

However, the vertical rules are not drawn in the blocks (created by \texttt{\Block}; cf. p. 4).

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{ninematrix} are compatible with \texttt{booktabs}.

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

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

\newcolumntype{I}{|!{\vrule}}
However, in this case, it is probably more clever to add a command \OnlyMainNiceMatrix (cf. p. 36): 
\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 \\
A&B&C&D \\hline
\end{tabular}

A B C D

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

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

A B C D

5.2 The thickness and the color of the rules

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

It’s well known that \texttt{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 \texttt{colortbl} is not loaded. For sake of compatibility, the command is also named \arrayrulecolor. The environments of \nicematrix also provide a key \texttt{rules/color} to fix the color of the rules in the current environment.

\begin{NiceTabular}{|ccc|}\[
\texttt{rules/color=[gray]{0.9},rules/width=1pt}
\]
\hline
rose & tulipe & lys \\
arum & iris & violette \\
muguet & dahlia & souci \\
\hline
\end{NiceTabular}

rose tulipe lys
arum iris violette
muguet dahlia souci

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 36.

5.3 The command \Hline

We have said that the vertical rules specified by a symbol “|” in the preamble (for the environments with preamble, of course) are not drawn in the blocks.\footnote{Those blocks are those created by the commands \Block and \multicolumn but also those implicitly determined by the continuous dotted lines (created by \Cdots, etc.).}

\textbf{New 5.13} In order to have horizontal rules with the same behaviour, \nicematrix provides the command \Hline.
5.4 The keys hlines and vlines

The key \texttt{hlines} draws all the horizontal rules and the key \texttt{vlines} draws all the vertical rules excepted in the blocks, created by \texttt{\textbackslash Block} and the virtual blocks determined by dotted lines: \texttt{\textbackslash Cdots}, \texttt{\textbackslash Vdots}, etc.. In fact, in the environments with delimiters (as \texttt{\{pNiceMatrix\}} or \texttt{\{bNiceArray\}}) the exterior 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}

New 5.13 It's possible to provide to the key \texttt{vlines} a (comma-separated) list of numbers, which are the numbers of the vertical rules to draw. The key \texttt{hlines} has a similar behaviour.

5.5 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 and excepted the rules corresponding of the frame of the blocks using the key \texttt{draw} which are drawn with their own characteristics).

\begin{NiceTabular}{cccc}
  \[hvlines, rules/color=blue\]
  rose & tulipe & marguerite & dahlia \\
  violette & \texttt{\textbackslash Block[draw=red]2-2}{\LARGE fleurs} & & souci \\
  pervenche & & & lys \\
  arum & iris & jacinthe & muguet
\end{NiceTabular}

5.6 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 \\
  & \texttt{\textbackslash Block[2-2]{B}} & & A \\
  & & A & A \\
  & 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.

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

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

\* The concept of corner is also used by the command `\arraycolor` in the `\CodeBefore` which takes in as option a key `except-corners`: cf. p. 14.

### 5.7 The command `\diagbox`

The command `\diagbox` (inspired by the package `diagbox`), allows, when it is used in a cell, to slash that cell diagonally downwards.\(^8\)

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

It’s possible to use the command `\diagbox` in a `\Block`.

### 5.8 Dotted rules

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

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

\*The author of this document considers that type of construction as graphically poor.
In the environments with an explicit preamble (like \{NiceTabular}, \{NiceArray}, etc.), it’s possible to draw a vertical dotted line with the specifier “:\n\left\begin{NiceArray}{cccccc} & & & & \\ 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{NiceArray}\right)

It’s possible to change in nicematrix the letter used to specify a vertical dotted line with the option letter-for-dotted-lines available in \texttt{\NiceMatrixOptions}. Thus released, the letter “:\n\hdottedline and “:\n\texttt{\hdotsfor} do likewise.

\section{The color of the rows and columns}

\subsection{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:

\begin{itemize}
\item The package colortbl patches array, leading to some incompatibilities (for instance with the command \hdotsfor).
\item 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.
  \begin{itemize}
  \item 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).
  \item A thin white line may appear between two cells of the same color. This phenomenon occurs when each cell is colored with its own instruction \texttt{fill} (the PostScript operator \texttt{fill} noted \texttt{f} in PDF). This is the case with colortbl: each cell is colored on its own, even when \texttt{\columncolor} or \texttt{\rowcolor} is used.
    \begin{itemize}
    \item As for this phenomenon, Adobe Reader gives better results than MuPDF.
    \end{itemize}
  \end{itemize}
\end{itemize}

The package nicematrix provides tools to avoid those problems.

\subsection{The tools of nicematrix in the \texttt{\CodeBefore}}

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

The extension nicematrix provides a key code-before for some code that will be executed before the drawing of the tabular.

\textbf{New 5.12} An alternative syntax is provided: it’s possible to put the content of that code-before between the keywords \texttt{\CodeBefore} and \texttt{\Body} at the beginning of the environment.

\footnote{However, one should remark that the package arydshln is not fully compatible with nicematrix.}

\footnote{In fact, with array, this is true only for \texttt{\hline} and “|” but not for \texttt{\cline}: cf p. 8}
New commands are available in that \CodeBefore: \cellcolor, \rectanglecolor, \rowcolor, \columncolor, \rowcolors, \chessboardcolors and \arraycolor.\footnote{Remark that, in the \CodeBefore, PGF/Tikz nodes of the form “(i-|j)” are also available to indicate the position to the potential rules: cf. p. 34.}

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

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

\begin{NiceTabular}{|c|c|c|}
\CodeBefore
\cellcolor[HTML]{FFFF88}{3-1,2-2,1-3}
\Body
\hline
a & b & c \\
\hline
e & f & g \\
\hline
h & i & j \\
\hline
\end{NiceTabular}

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

\begin{NiceTabular}{|c|c|c|}
\CodeBefore
\rectanglecolor{blue!15}{2-2}{3-3}
\Body
\hline
a & b & c \\
\hline
e & f & g \\
\hline
h & i & j \\
\hline
\end{NiceTabular}

- The command \rowcolor takes its name from the command \rowcolor of colorbl. 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}–\textit{b} represent all the rows from the row \textit{a} until the end).

\begin{NiceTabular}{|c|c|c|}
\CodeBefore
\rowcolor{red}{3-2,1-3}
\Body
\hline
a & b & c \\
\hline
e & f & g \\
\hline
h & i & j \\
\hline
\end{NiceTabular}
\begin{NiceArray}{lll} \CodeBefore \rowcolor{red!15}{1,3-5,8-} \Body \rowcolor{red!15}{1,3-5,8-} \begin{NiceArray}{lll} 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} \end{NiceArray}$

- The command \texttt{\rowcolors} takes its name from the command \texttt{\rowcolor} 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 with the two colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

In fact, the first (mandatory) argument is, more generally, a comma separated list of intervals describing the rows involved in the action of \texttt{\rowcolors} (an interval of the form \texttt{i-j} describes in fact the interval of all the rows of the tabular, beginning with the row \texttt{i}).

The \texttt{\rowcolors} is an optional list of pairs key-value (the optional argument in the first position corresponds to the colorimetric space). The available keys are \texttt{cols}, \texttt{restart} and \texttt{respect-blocks}.

- The key \texttt{cols} describes a set of columns. The command \texttt{\rowcolors} will color only the cells of these columns. The value is a comma-separated list of intervals of the form \texttt{i-j}.
- With the key \texttt{restart}, each interval of rows (specified by the first mandatory argument) begins with the same color.\footnote{The command \texttt{\rowcolors} of \texttt{xcolor} is available when \texttt{xcolor} is loaded with the option \texttt{table}. That option also loads the package \texttt{colortbl}.}
- With the key \texttt{respect-blocks} the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command \texttt{\Block}; cf. p. 4).

\begin{NiceTabular}{clr} \CodeBefore \rowcolors{2}{blue!10}{}[cols=2-3,restart] \Body \Block{1-+}{Results} \begin{NiceArray}{ll} A & John 12 \\ & Stephen 8 \\ & Sarah 18 \\ B & Ashley 20 \\ & Henry 14 \\ & Madison 15 \end{NiceArray} \end{NiceTabular}

\footnote{Otherwise, the color of a given row relies only upon the parity of its number.}
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]
\CodeBefore
\chessboardcolors{red!15}{blue!15}
\Body
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1
\end{pNiceMatrix}

We have used the key \texttt{r} which aligns all the columns rightwards (cf. p. 30).

**New 5.13** The command \texttt{\arraycolor} takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 17). This command is useful thanks to its key \texttt{except-corners} (in the third argument, which is an optional argument between square brackets). The definition of these “corners” has been given p. 9 when we have presented the key \texttt{hvlines-except-corners}.

\begin{NiceTabular}{*{6}{c}}[cell-space-top-limit=3pt]
\CodeBefore
\arraycolor{blue!10}[except-corners=NE]
\Body
1 \\
1 & 1 \\
1 & 2 & 1 \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
1 & 5 & 10 & 10 & 5 & 1
\end{NiceTabular}

One should remark that these commands are compatible with the commands of \texttt{booktabs} (\texttt{\toprule}, \texttt{\midrule}, \texttt{\bottomrule}, etc). However, \texttt{booktabs} is not loaded by \texttt{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 for now, this key is not available in \texttt{NiceMatrixOptions}.}

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

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

\begin{Verbatim}
\NewDocumentCommand { \Blue } { } { \columncolor{blue!15} }
\end{Verbatim}

\begin{NiceTabular}[colortbl-like]{>{\Blue}c>{\Blue}cc}
\toprule
\rowcolor{red!15} Last name & First name & Birth day \ \\
Achard & Jacques & 5 juin 1962 \ \\
Lefebvre & Mathilde & 23 mai 1988 \ \\
Vanesse & Stephany & 30 octobre 1994 \ \\
Dupont & Chantal & 15 janvier 1998 \ \\
\bottomrule
\end{NiceTabular}
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}{|Cc|c|c|}
\hline
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo \\
\hline
\end{NiceTabular}

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

$\begin{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 option \texttt{columns-width}: all the columns of the array will have a width equal to the widest cell of the array.\footnote{The result is achieved with only one compilation (but PGF/Tikz will have written informations in the .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}.

\\NiceMatrixOptions\[columns-width=10mm\]

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 \texttt{\NiceMatrixBlock} with the option \texttt{auto-columns-width}\footnote{At this time, this is the only usage of the environment \texttt{\NiceMatrixBlock} but it may have other usages in the future.}. The environment \texttt{\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} \quad \begin{bNiceMatrix}
1 & 1245345 \\
345 & 2
\end{bNiceMatrix}$

15 The result is achieved with only one compilation (but PGF/Tikz will have written informations in the .aux file and a message requiring a second compilation will appear).
The exterior rows and columns

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

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

\begin{pNiceMatrix}[\firstrow,\lastrow,\firstcol,\lastcol,\nullify-dots]

\begin{align*}
\begin{array}{cccc}
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 \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
\end{array}
\end{align*}
\end{pNiceMatrix}

The dotted lines have been drawn with the tools presented p. 18.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. \texttt{NiceTabular}, \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 \texttt{r} for the first column and \texttt{l} for the last one.\footnote{The users wishing exteriors columns with another type of alignment should consider the command \texttt{\SubMatrix} available in the \texttt{\CodeAfter} (cf. p. 24).}

- One may wonder how \texttt{nicematrix} determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
  - For the environments with explicit preamble, like \texttt{NiceTabular} and \texttt{pNiceArray}, the number of columns can obviously be computed from the preamble.
  - When the option \texttt{light-syntax} (cf. p. 32) is used, \texttt{nicematrix} has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option \texttt{light-syntax}). The analysis of this whole body gives the number of rows (but not the number of columns).
  - In the other cases, \texttt{nicematrix} compute the number of rows and columns during the first compilation and write the result in the \texttt{aux} file for the next run.

  \textit{However, it’s possible to provide the number of the last row and the number of the last column as values of the options \texttt{last-row} and \texttt{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 \texttt{code-for-first-row}, \texttt{code-for-last-row, code-for-first-col} and \texttt{code-for-last-col}. These options specify tokens that will be inserted before each cell of the corresponding row or column.

\begin{verbatim}
\NiceMatrixOptions{code-for-first-row = \color{red},
                 code-for-first-col = \color{blue},
                 code-for-last-row = \color{green},
                 code-for-last-col = \color{magenta}}
\end{verbatim}
\begin{NiceArray}{cc|cc}
\begin{array}{cccc}
\hline
C_1 & \Cdots & & C_4 \\
\hline
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 \\
\end{array}
\end{NiceArray}$

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 36.

- 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. 16) 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. If you are looking for a workaround, consider the command \SubMatrix in the \CodeAfter described p. 24.

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 to change the color of these lines with the option color. \footnote{The command \Iddots, defined in nicematrix, is a variant of \iddots with dots going forward. If mathdots is loaded, the version of mathdots is used. It corresponds to the command \adots of unicode-math.}

\begin{bNiceMatrix}
a_1 & \Cdots & & a_1 \\
\Vdots & a_2 & \Cdots & & a_2 \\
& a_3 & \Ddots & & a_3 \\
\end{bNiceMatrix}$

\begin{align*}
\begin{bNiceMatrix}
a_1 & \cdots & a_1 \\
\vdots & \cdots & \vdots \\
a_1 & a_2 & a_n \\
\end{bNiceMatrix}
\end{align*}$

\begin{align*}
\begin{bNiceMatrix}
a_1 & \cdots & a_1 \\
\vdots & \cdots & \vdots \\
& a_2 \cdots a_n \\
\end{bNiceMatrix}
\end{align*}$

\footnote{The precise definition of a “non-empty cell” is given below (cf. p. 37).

It’s also possible to change the color of all these dotted lines with the option xdots/color (xdots to remind that it works for \Cdots, \Ldots, \Vdots, etc.): cf. p. 22.}
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 \\
\[1cm\]
0 & \Cdots & \Hspace*{1cm} & 0
\end{bNiceMatrix}

In nicematrix, one should use \Hspace* and not \hspace for such an usage because nicematrix loads array. One may also remark that it’s possible to fix the width of a column by using the environment \NiceArray (or one of its variants) with a column of type v or W: see p. 16.

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.

\begin{pNiceMatrix}
h & i & j & k & l & m \\ x & \Ldots & \Ldots & \Ldots & \Ldots & x \\
\end{pNiceMatrix}

$C = \begin{pmatrix} h & i & j & k & l & m \\ x & \Ldots & \Ldots & \Ldots & \Ldots & x \end{pmatrix}$

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).

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

$D = \begin{pmatrix} h & i & j & k & l & m \\ x & \Ldots & \ & \ & \ & x \end{pmatrix}$

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 \texttt{amsmath} in order to draw horizontal dotted lines in a matrix. In the environments of \nicematrix, one should use instead \Hdotsfor in order to draw dotted lines similar to the other dotted lines drawn by the package \nicematrix.

As with the other commands of \nicematrix (like \Cdots, \Ldots, \Vdots, etc.), the dotted line drawn with \Hdotsfor extends until the contents of the cells on both sides.

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

$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & \hdotsfor{3} & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$

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

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

$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & \hdotsfor{3} \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$

Remark: Unlike the command \hdotsfor of \texttt{amsmath}, the command \Hdotsfor may be used even when the package \texttt{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{pNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n] \\
& \hspace*{20mm} & \Cdots & \C[a_1,a_n]{(p)} \& \Cdots & \C[a_1,a_n]{(p)} \\
\Vdots & \Ddots & \Vdots & \Vdotsfor{1} & \Ddots & \Vdotsfor{1} & \Vdotsfor{1} \\
C[a_n,a_1] & \Cdots & \C[a_n,a_n] \\
& \C[a_n,a_n]{(p)} \& \Cdots & \C[a_n,a_n]{(p)} \\
\rule{0pt}{15mm}\NotEmpty & \Vdotsfor{1} & \Ddots & \Vdotsfor{1} & \Vdotsfor{1} & \Vdotsfor{1} & \Vdotsfor{1} \\
C[a_1^{(p)},a_1] & \Cdots & C[a_1^{(p)},a_n]
\end{pNiceMatrix}\[22\text{We recall that when xcolor is loaded with the option table, the package colortbl is loaded.}]

20
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`.

- 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 nicematrix.

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

9.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `{CodeAfter}` which is described p. 23) 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{Verbatim}
23The 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.
\end{Verbatim}
9.5 Customisation of the dotted lines

The dotted lines drawn by \Ldots, \Cdots, \Vdots, \Ddots, \Idots and \Hdots (and by the command \line in the \CodeAfter which is described p. 23) 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, as options of \CodeAfter 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. 17.

The option xdots/shorten

The option xdots/shorten fixes the margin of both extremities of the line. The name is derived from the options “shorten >” and “shorten <” of Tikz but one should notice that nicematrix only provides xdots/shorten. The initial value of this parameter is 0.3 em (it is recommended to use a unit of length dependent of the current font).

The option xdots/line-style

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

\texttt{\tikz \draw [dotted] (0,0) -- (5,0);}

In order to provide lines with rounded dots in the style of those provided by \ldots (at least with the Computer Modern fonts), the package nicematrix embeds its own system to draw a dotted line (and this system uses PGF and not Tikz). This style is called standard and that’s the initial value of the parameter xdots/line-style.

However (when Tikz is loaded) it’s possible to use for xdots/line-style any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tizk pathes (with the exception of “color”, “shorten >” and “shorten <”).
Here is for example a tridiagonal matrix with the style *loosely dotted*:

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

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

9.6 The dotted lines and the rules

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

\[
\begin{bNiceMatrix}[margin,hvlines]
\Block{3-3}<\LARGE>{A} & 0 \\ \Ddots & \Ddots & \Ddots & b \\ \Vdots & \Vdots & \Vdots & \Vdots & 0 \\ 0 & \Cdots & 0 & 0
\end{bNiceMatrix}
\]

10 The \texttt{\textbackslash CodeAfter}

The option \texttt{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 \texttt{code-before} described p. 12.}

For the legibility of the code, an alternative syntax is provided: it’s possible to give the instructions of the \texttt{\textbackslash CodeAfter} at the end of the environment, after the keyword \texttt{\textbackslash CodeAfter}. Although \texttt{\textbackslash CodeAfter} is a keyword, it takes in an optional argument (between square brackets). The keys accepted form a subset of the keys of the command \texttt{\textbackslash WithArrowsOptions}.

The experienced users may use, for instance, the PGF/Tikz nodes created by \texttt{nicematrix} in the \texttt{\textbackslash CodeAfter}. These nodes are described further beginning on p. 32.

Moreover, two special commands are available in the \texttt{\textbackslash CodeAfter}: \texttt{line} and \texttt{SubMatrix}.

10.1 The command \texttt{line} in the \texttt{\textbackslash CodeAfter}

The command \texttt{\textbackslash line} draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form \texttt{i-j} where \texttt{i} is the number of the row and \texttt{j} is the number of the column. It may be used, for example, to draw a dotted line between two adjacent cells.
The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 22).

10.2 The command `\SubMatrix` in the `\CodeAfter`

The command `\SubMatrix` provides a way to put delimiters on a portion of the array considered as a submatrix. The command `\SubMatrix` takes in five arguments:

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX: `, `\[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `\.`;
- the second argument is the upper-left corner of the submatrix with the syntax `i-j` where `i` the number of row and `j` the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of key-value pairs.

One should remark that the command `\SubMatrix` draws the delimiters after the construction of the array: no space is inserted by the command `\SubMatrix` itself. That’s why, in the following example, we have used the key `margin` and you have added by hand some space between the third and fourth column with `@{\hspace{1.5em}}` in the preamble of the array.

\[
\begin{NiceArray}{ccc@{\hspace{1.5em}}c}
\begin{pNiceMatrix}
\CodeAfter \line{2-2}{3-3}
\end{pNiceMatrix}\end{NiceArray}
\]

The options of the command `\SubMatrix` are as follows:

- `left-xshift` and `right-shift` shift horizontally the delimiters (there exists also the key `xshift` which fixes both parameters);
- `extra-height` adds a quantity to the total height of the delimiters (height `\ht` + depth `\dp`);
- `delimiters/color` fixes the color of the delimiters (also available in `\NiceMatrixOptions`, in the environments with delimiters and as option of the keyword `\CodeAfter`);
- `slim` is a boolean key: when that key is in force, the horizontal position of the delimiters is computed by using only the contents of the cells of the submatrix whereas, in the general case, the position is computed by taking into account the cells of the whole columns implied in the submatrix (see example below).
- `vlines` contents a list of numbers of vertical rules that will be drawn in the sub-matrix (if this key is used without value, all the vertical rules of the sub-matrix are drawn).

\[\begin{pmatrix}
I & 0 & \cdots & 0 \\
0 & I & \cdots & \vdots \\
\vdots & \ddots & I & 0 \\
0 & \cdots & 0 & I
\end{pmatrix}\]

There is no optional argument between square brackets in first position because a square bracket just after `\SubMatrix` must be interpreted as the first (mandatory) argument of the command `\SubMatrix`: that bracket is the left delimiter of the sub-matrix to construct (e.g.: `\SubMatrix[{2-2}{4-7}]`).
• **hlines** is similar to **vlines** but for the horizontal rules;
• **h\(v\)lines**, which must be used without value, draws all the vertical and horizontal rules.

One should remark that these keys add their rules after the construction of the main matrix: no space is added between the rows and the columns of the array for these rules.

All these keys are also available in \texttt{\NiceMatrixOptions}, at the level of the environments of \texttt{nicematrix} or as option of the command \texttt{\CodeAfter} with the prefix \texttt{sub-matrix} which means that their names are therefore \texttt{sub-matrix/left-xshift}, \texttt{sub-matrix/right-xshift}, \texttt{sub-matrix/xshift}, etc.

\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]
& & \frac{1}{2} \\
& & \frac{1}{4} \\
a & b & \frac{1}{2}a + \frac{1}{4}b \\
c & d & \frac{1}{2}c + \frac{1}{4}d \CodeAfter
\SubMatrix(\{1-3\}\{2-3\})
\SubMatrix(\{3-1\}\{4-2\})
\SubMatrix(\{3-3\}\{4-3\})\end{NiceArray}

Here is the same example with the key \texttt{slim} used for one of the submatrices.

\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]
& & \frac{1}{2} \\
& & \frac{1}{4} \\
a & b & \frac{1}{2}a + \frac{1}{4}b \\
c & d & \frac{1}{2}c + \frac{1}{4}d \CodeAfter
\SubMatrix(\{1-3\}\{2-3\})[slim]
\SubMatrix(\{3-1\}\{4-2\})
\SubMatrix(\{3-3\}\{4-3\})\end{NiceArray}

There is also a key \texttt{name} which gives a name to the submatrix created by \texttt{\SubMatrix}. That name is used to create PGF/Tikz nodes: cf p. 35.

### 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 preferentially. 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}.  

25
11.2 The notes of tabular

The package nicematrix also provides a command \texttt{\tabularnote} which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say \{\texttt{NiceTabular}\}, \{\texttt{NiceArray}\} and \{\texttt{NiceMatrix}\}. In fact, this command is available only if the extension \texttt{enumitem} has been loaded (before or after \texttt{nicematrix}). Indeed, the notes are composed at the end of the array with a type of list provided by the package \texttt{enumitem}.

\begin{NiceTabular}{@{}llr@{}}
\toprule
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.} & Jacques & June 5, 2005 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.} & Mathilde & January 23, 1975 \\
Vanesse & Stephany & October 30, 1994 \\
Dupont & Chantal & January 15, 1998 \\
\bottomrule
\end{NiceTabular}

- If you have several successive commands \texttt{\tabularnote{...}} \textit{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{\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}).
- There is a key \texttt{tabularnote} which provides a way to insert some text in the zone of the notes before the numbered tabular notes.
- 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{\tabularnote} may be used \textit{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{\tabularnote} (with the usual command \texttt{\label} used after the \texttt{\tabularnote}).

For an illustration of some of those remarks, see table 1, p. 27. This table has been composed with the following code.
Table 1: Use of \texttt{\textbackslash tabularnote}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</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>

Some text before the notes.

\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.

### 11.3 Customisation of the tabular notes

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

- \texttt{notes/para}
- \texttt{notes/bottomrule}
- \texttt{notes/style}
- \texttt{notes/label-in-tabular}
- \texttt{notes/label-in-list}
- \texttt{notes/enumitem-keys}
- \texttt{notes/enumitem-keys-para}
- \texttt{notes/code-before}

For sake of commodity, it is also possible to set these keys in \texttt{\NiceMatrixOptions} via a key \texttt{notes} which takes in as value a list of pairs \texttt{key=value} where the name of the keys need no longer be prefixed by \texttt{notes}.
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{\textup{a}}\texttt{ph}{#1}`
  Another possible value should be a mere `\texttt{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:

  \begin{verbatim}
  \NiceMatrixOptions{notes/label-in-tabular = \textsuperscript{#1}}
  \end{verbatim}

- 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:

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

  The command `\texttt{nobreak}` is for the event that the option `para` is used.
• The notes are composed at the end of the tabular by using internally a style of list of \textsc{enumitem}.
  
  The key \texttt{notes/enumitem-keys} specifies a list of pairs \texttt{key=value} (following the specifications of \texttt{enumitem}) to customize that type of list.
  
  Initial value: \texttt{noitemsep, leftmargin = *, align = left, labelsep = Opt}
  
  This initial value contains the specification \texttt{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. 27).

• The key \texttt{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 \texttt{key=value} should correspond to such a list of type \texttt{inline}.
  
  Initial value: \texttt{afterlabel = \nobreak, itemjoin = \quad}

• The key \texttt{notes/code-before} is a token list inserted by \texttt{nicematrix} just before the composition of the notes at the end of the tabular.
  
  Initial value: \texttt{empty}
  
  For example, if one wishes to compose all the notes in gray and \texttt{\footnotesize}, he should use that key:

\begin{verbatim}
\NiceMatrixOptions{notes/code-before = \footnotesize \color{gray}}
\end{verbatim}

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 customisation of the tabular notes, see p. 38.

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

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

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

12 Other features

12.1 Use of the column type \texttt{S} of \texttt{siunitx}

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

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

On the other hand, the \texttt{d} columns of the package \texttt{dcolumn} are not supported by \texttt{nicematrix}.

29
12.2 Alignment option in \{NiceMatrix\}

The environments without preamble (\{NiceMatrix\}, \{pNiceMatrix\}, \{bNiceMatrix\}, etc.) provide two options \texttt{l} and \texttt{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 \texttt{\rotate}

The package \texttt{nicematrix} provides a command \texttt{\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 sens.

In the following command, we use that command in the \texttt{code-for-first-row}.

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

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

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

12.4 The option \texttt{small}

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

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

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

In fact, the option \texttt{small} corresponds to the following tuning: 

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

### 12.5 The counters iRow and jCol 

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

In the \texttt{code-before} (cf. p. 12) and in the \texttt{CodeAfter} (cf. p. 23), \texttt{iRow} represents the total number of rows (excepted the potential exterior rows) and \texttt{jCol} represents the total number of columns (excepted the potential exterior columns). 

\begin{verbatim}
$\begin{pNiceMatrix}% don't forget the % 
  \[first-row, \ 
  first-col, \ 
  \text{code-for-first-row} = \mathbf{\text{\texttt{alph}[jCol]}}, \ 
  \text{code-for-first-col} = \mathbf{\text{\texttt{arabic}[iRow]}} \] 
\end{verbatim}$

\begin{verbatim}
\begin{array}{r r r r}
  & a & b & c & d \\
  k & k & k & 1 & 2 & 3 & 4 \\
  k & 5 & 6 & 7 & 8 \\
  k & 9 & 10 & 11 & 12 \\
\end{array}
\end{verbatim}

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

The package \texttt{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 VAutoNiceMatrix} and \texttt{\textbackslash BAutoNiceMatrix}. 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). 

\begin{verbatim}
$C = \pAutoNiceMatrix\{3-3\}{C_{\arabic{iRow},\arabic{jCol}}}$
\end{verbatim}

\begin{verbatim}
C = \begin{pmatrix}
  C_{1,1} & C_{1,2} & C_{1,3} \\
  C_{2,1} & C_{2,2} & C_{2,3} \\
  C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix}
\end{verbatim}
12.6 The option light-syntax

The option `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.

\[
\begin{bNiceMatrix}
\text{light-syntax}, \text{first-row}, \text{first-col}
\{ \}
\begin{array}{cc}
a & b \\
2 \cos a & \{ \cos a + \cos b \} \\
a & 2 \cos a \\
\cos a + \cos b & \{ 2 \cos b \}
\end{array}
\end{bNiceMatrix}
\]

When this option is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.\(^2\)

12.7 Color of the delimiters

For the environments with delimiters (\texttt{pNiceArray}, \texttt{pNiceMatrix}, etc.), it’s possible to change the color of the delimiters with the key `delimiters/color`.\(^2\)

\[
\begin{bNiceMatrix}[delimiters/color=red]
1 & 2 \\
3 & 4
\end{bNiceMatrix}
\]

12.8 The environment \texttt{NiceArrayWithDelims}

In fact, the environment \texttt{pNiceArray} and its variants are based upon a more general environment, called \texttt{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 \texttt{NiceArrayWithDelims} if we want to use atypical or asymmetrical delimiters.

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

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).

\textbf{Caution} : By default, no node is created in a empty cell. However, it’s possible to impose the creation of a node with the command `\NotEmpty`.\(^3\)

\(^2\)The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn’t behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.

\(^2\)`delimiters-color` is a synonymous (deprecated) for `delimiters/color`.

\(^3\)One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 18).
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 a internal global counter.
In the environment with the number \( n \), the node of the row \( i \) and column \( j \) has for name \( \text{nm-}n-i-j \). The command \texttt{\textbackslash 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 “\texttt{name-}i-j” where \texttt{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. In the following examples, we assume that Tikz has been loaded.

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

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

In the \texttt{\CodeAfter}, the things are easier : one must refer to the nodes with the form \( i-j \) (we don’t have 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. 44).

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

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 \texttt{create-medium-nodes} and the second ones with the option \texttt{create-large-nodes}.

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{pNiceMatrix}
(a & a+b & a+b+c) \\
(a & a & a+b) \\
(a & a & a)
\end{pNiceMatrix}

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.

\begin{pNiceMatrix}
(a & a+b & a+b+c) \\
(a & a & a+b) \\
(a & a & a)
\end{pNiceMatrix}

\footnote{There is also an option \texttt{create-extra-nodes} which is an alias for the conjunction of \texttt{create-medium-nodes} and \texttt{create-large-nodes}.}

\footnote{There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 17).}
The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options \texttt{left-margin} and \texttt{right-margin} to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options \texttt{left-margin} and \texttt{right-margin}.

\[
\begin{array}{ccc}
\quad & a & a + b \\
\quad & a & a + b \\
\quad & a & a + b \\
\end{array}
\]

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

\[
\begin{array}{ccc}
\quad & a & a + b \\
\quad & a & a + b \\
\quad & a & a + b \\
\end{array}
\]

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

Here is an array composed with the following code:

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

Here, we have colored all the cells of the array with \texttt{chessboardcolors}.

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

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

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

\textbf{13.3 The nodes which indicate the position of the rules}

\textbf{New 5.11} The package \texttt{nicematrix} creates a PGF/Tikz node merely called \texttt{i} (with the classical prefix) at the intersection of the horizontal rule of number \texttt{i} and the vertical rule of number \texttt{j} (more specifically the potential position of those rules because maybe there are not actually drawn). These nodes are available in the \texttt{code-before} and the \texttt{\CodeAfter}.

\textbf{New 5.13} The last node has also an alias called \texttt{last}.

\footnote{The options \texttt{left-margin} and \texttt{right-margin} take dimensions as values but, if no value is given, the default value is used, which is \texttt{\arraycolsep} (by default: 5 pt). There is also an option \texttt{margin} to fix both \texttt{left-margin} and \texttt{right-margin} to the same value.}
If we use Tikz (we remind that nicematrix does not load Tikz by default, by only PGF, which is a sub-layer of Tikz), we can access, in the \CodeAfter but also in the code-before, to the intersection of the (potential) horizontal rule $i$ and the (potential) vertical rule $j$ with the syntax $(i\mid j)$.

\begin{NiceMatrix}
\CodeBefore
  \tikz \draw [fill=red!15] (7-|4) |- (8-|5) |- (9-|6) |- cycle ;
\Body
  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}

\section{The nodes corresponding to the command \texttt{\SubMatrix}}

The command \texttt{\SubMatrix} available in the \CodeAfter has been described p. 24.

If a command \texttt{\SubMatrix} has been used with the key \texttt{name} with an expression such as \texttt{name=MyName} three PGF/Tikz nodes are created with the names MyName-left, MyName and MyName-right.

The nodes MyName-left and MyName-right correspond to the delimiters left and right and the node MyName correspond to the submatrix itself.

In the following example, we have highlighted these nodes (the submatrix itself has been created with \texttt{\SubMatrix\{(2-2\{3-3\)}}).

\begin{equation*}
\begin{pmatrix}
  1 & 21 & 23 & 345 & 345 \\
  45 & 346 & 863 & 444 \\
  3462 & 38458 & 34 & 294 \\
  34 & 7 & 78 & 309
\end{pmatrix}
\end{equation*}

\section{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{\_nicematrix} is public and each variable with name beginning with \texttt{\_{\_nicematrix}} is private.}:
• \g_nicematrix_code_before_tl;
• \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 \g_nicematrix_code_before_tl needs one compilation more (because the instructions are written on the aux file to be used during the next run).

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

```
\ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatch:nnn
  {\tikz \fill [ pattern = north-west~lines , pattern~color = #3 ]
  ( #1 -| #2) rectangle ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;}
\NewDocumentCommand \hatchcell { ! O { black } }{
  \tl_gput_right:Nx \g_nicematrix_code_before_tl { \__pantigny_hatch:nnn { \arabic{iRow} } { \arabic{jCol} } { #1 } }
}\ExplSyntaxOff
```

Here is an example of use:

```
\begin{NiceTabular}{ccc}[hvlines]
Tokyo & Paris & London \\
Lima & \hatchcell[blue!30]Oslo & Miami \\
Los Angeles & Madrid & Roma
\end{NiceTabular}
```

15  Technical remarks

15.1  Definition of new column types

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

For example, one may wish to define a new column type ? in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job:\footnote{The command \vrule is a TeX (and not LaTeX) command.}

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

The heavy vertical rule won’t extend in the exterior rows.\footnote{Of course, such rule is defined by the classical technics of nicematrix and, for this reason, won’t cross the double rules of \hline.}
This specifier \texttt{?} 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{37} 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}
$A = \begin{pNiceMatrix}
1 & \Cdots & & 1 \\
\vdots & \Ddots & & \vdots \\
a+b & \ & \ & 1 \\
\end{pNiceMatrix}$
\end{verbatim}

\begin{verbatim}
$A = \begin{pNiceMatrix}
1 & \Cdots & & 1 \\
\vdots & \Ddots & & \vdots \\
a+b & \ & \ & 1 \\
\end{pNiceMatrix}$
\end{verbatim}

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

**The same example without parallelization:**

\begin{verbatim}
$A = \begin{pNiceMatrix}
1 & \Cdots & & 1 \\
\vdots & \Ddots & & \vdots \\
\end{verbatim}

It’s possible to specify the instruction \texttt{\Ddots} which will be drawn first (and which will be used to draw the other diagonal dotted lines when the parallelization is in force) with the key \texttt{draw-first}: \texttt{\Ddots\[draw-first\]}.

15.3 The “empty” cells

An instruction like \texttt{\Ldots}, \texttt{\Cdots}, etc. tries to determine the first non-empty cell 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 \texttt{&}). The precise rules are as follow.

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

\begin{verbatim}
$\begin{pNiceMatrix}
a & b \\
c \\
\end{pNiceMatrix}$
\end{verbatim}

\textsuperscript{37}\textbf{Note:} We speak of the lines created by \texttt{\Ddots} and not the lines created by a command \texttt{\line} in code-after.
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 containing the command `\NotEmpty` is not empty (and a PGF/Tikz node) is created in that cell.

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

15.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea\(^{38}\). The environment `{matrix}` of `amsmath` and its variants `{pmatrix}`, `{vmatrix}`, etc. of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`,\(^{39}\) The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of `array` (for example, when adapting an existing document) it’s possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

15.5 Incompatibilities

The package `nicematrix` is not fully compatible with the package `arydshln` (because this package redefines many internal of `array`). Anyway, in order to use `arydshln`, one must first free the letter “;” by giving a new letter for the vertical dotted rules of `nicematrix`:

```
\NiceMatrixOptions{letter-for-dotted-lines=;}
```

As for now, the package `nicematrix` is not compatible with `aastex63`. If you want to use `nicematrix` with `aastex63`, send me an email and I will try to solve the incompatibilities.

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. 25.

Let’s consider that we wish to number the notes of a tabular with stars.\(^{40}\)

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument.\(^{41}\)

---

\(^{38}\)In the documentation of `{amsmath}`, we can read: *The extra space of `\arraycolsep` that `array` adds on each side is a waste so we remove it in `{matrix}` (perhaps we should instead remove it from `array` in general, but that’s a harder task).*

\(^{39}\)And not by inserting `@{}` on both sides of the preamble of the array. As a consequence, the length of the `\hline` is not modified and may appear too long, in particular when using square brackets.

\(^{40}\)Of course, it’s realistic only when there is very few notes in the tabular.

\(^{41}\)In fact: the value of its argument.
Of course, we change the style of the labels with the key 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 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 value(tabularnote) (because tabularnote is the LaTeX counter used by \tabularnote and, therefore, at the end of the tabular, its value is equal to the total number of tabular notes). We use the key widest* of enumitem in order to require a width equal to that value: widest*={value(tabularnote)}.

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

\begin{NiceTabular}{{}llr{}}[first-row,code-for-first-row = \bfseries]
\toprule
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.} & Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.} & Mathilde & 23 mai 1988 \\
Vannes & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

<table>
<thead>
<tr>
<th>Last name</th>
<th>First name</th>
<th>Birth day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achard*</td>
<td>Jacques</td>
<td>June 5, 2005</td>
</tr>
<tr>
<td>Lefebvre**</td>
<td>Mathilde</td>
<td>January 23, 1975</td>
</tr>
<tr>
<td>Vanesse</td>
<td>Stephany</td>
<td>October 30, 1994</td>
</tr>
<tr>
<td>Dupont</td>
<td>Chantal</td>
<td>January 15, 1998</td>
</tr>
</tbody>
</table>

*Achard is an old family of the Poitou.
**The name Lefebvre is an alteration of the name Lefebure.
16.2 Dotted lines

An example with the resultant of two polynomials:

\[
\begin{vNiceArray}{cccc:ccc}[columns-width=6mm]
\begin{array}{ccccccc}
\hline
a_0 & && &b_0 & & \\
\hline
a_1 & & & &b_1 & & \\
\hline
\vdots & & & &\vdots & & \\
\hline
a_p & & &a_0 & & &b_1 \\
& & &a_1 &b_q & & \vdots \\
& & &\vdots & & &\ddots \\
& & &a_p & & &b_q \\
\end{array}
\end{vNiceArray}
\]

An example for a linear system:

\[
\begin{pNiceArray}{*6c|c}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
\begin{array}{ccccccc}
\hline
1 & 1 & 1 &\Cdots & & 1 & 0 & \\
0 & 1 & 0 &\Cdots & & 0 & & L_2 \gets L_2-L_1 \\
0 & 0 & 1 &\Cdots & & \vdots & & L_3 \gets L_3-L_1 \\
& & &\ddots & & &\ddots \\
0 & & &\Cdots & 0 & 1 & \cdots \\
\end{array}
\end{pNiceArray}
\]

16.3 Dotted lines which are no longer dotted

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

\NiceMatrixOptions{code-for-first-row = \scriptstyle,code-for-first-col = \scriptstyle }
\setcounter{MaxMatrixCols}{12}
\newcommand{\blue}{\color{blue}}
\[
\begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
\begin{array}{ccccccc}
\hline
1 & 1 & 1 &\Cdots & & 1 & 0 \\
0 & 1 & 0 &\Cdots & & 0 & \cdots L_2\gets L_2-L_1 \\
0 & 0 & 1 &\Cdots & & 0 & \cdots L_3\gets L_3-L_1 \\
& & &\ddots & & &\ddots \\
0 & & &\Cdots & 0 & 1 & \cdots \cdots L_n\gets L_n-L_1 \\
\end{array}
\end{pNiceMatrix}\]

1 & & \Dots & & & & \Dots \\\n& \Dots [,line-style=standard] \\\n& & 1 \\\n& \Cdots [,color=blue,line-style=dashed] & & 0 &
& \Cdots & & \blue 1 & & \Cdots & \blue \leftarrow i \\\n& & 1 \\\n& \Cdots & & \Dots [,line-style=standard] & & \Dots \\\n& & \blue 1 & & \Cdots & \blue 0 & & \Cdots & \blue \leftarrow j \\\n& & 1 \\\n& \Cdots & & \Dots [,line-style=standard] \\\n& & \Dots & & \blue \overset{\uparrow}{i} & & \blue \overset{\uparrow}{j} \\\n& \end{pNiceMatrix}\]

\begin{bNiceMatrix}[first-row,first-col]
& & \Ldots[,line-style={solid,<->},shorten=0pt]^{n \text{ columns}} \\\n1 & 1 & 1 & \Ldots & 1 \\\n1 & 1 & 1 & & 1 \\\n\Vdots[,line-style={solid,<->}]_{n \text{ rows}} & 1 & 1 & 1 & & 1 \\\n1 & 1 & 1 & & 1 \\\n1 & 1 & 1 & \Ldots & 1
\end{bNiceMatrix}

In fact, it’s even possible to draw solid lines with the commands \Cdots, \Dots, etc.

\NiceMatrixOptions
{nullify-dots,code-for-first-col = \color{blue},code-for-first-col=\color{blue}}
$\begin{pNiceMatrix}[first-row,first-col]$
& & \Ldots[,line-style={solid,<->},shorten=0pt]^{n \text{ columns}} \\\n& 1 & 1 & 1 & & 1 \\\n& 1 & 1 & 1 & & 1 \\\n\Vdots[,line-style={solid,<->}]_{n \text{ rows}} & 1 & 1 & 1 & & 1 \\\n& 1 & 1 & 1 & & 1 \\\n& 1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$

16.4 Stacks of matrices

We often need to compose mathematical matrices on top on each other (for example for the resolution of linear systems).

In order to have the columns aligned one above the other, it’s possible to fix a width for all the columns. That’s what is done in the following example with the environment \texttt{NiceMatrixBlock} and its option \texttt{auto-columns-width}.
$\begin{pNiceArray}{rrrr|r}
  12 & -8 & 7 & 5 & 3 \\
  3 & -18 & 12 & 1 & 4 \\
 -3 & -46 & 29 & -2 & -15 \\
  9 & 10 & -5 & 4 & 7
\end{pNiceArray}$

$\begin{pNiceArray}{rrrr|r}
  12 & -8 & 7 & 5 & 3 \\
  0 & 64 & -41 & 1 & 19 { L_2 \gets L_1-4L_2 } \\
  0 & -192 & 123 & -3 & -57 { L_3 \gets L_1+4L_3 } \\
  0 & -64 & 41 & -1 & -19 { L_4 \gets 3L_1-4L_4 } 
\end{pNiceArray}$

$\begin{pNiceArray}{rrrr|r}
  12 & -8 & 7 & 5 & 3 \\
  0 & 64 & -41 & 1 & 19 \\
  0 & 0 & 0 & 0 & 0 { L_3 \gets 3 L_2 + L_3 }
\end{pNiceArray}$

However, one can see that the last matrix is not perfectly aligned with the other. That’s why, in LaTeX, the parenthesis have not exactly the same width (smaller parenthesis are a bit slimer).

\begin{itemize}
  \item In order the solve that problem, it’s possible to require the delimiters to be composed with the maximal width, thanks to the boolean key `delimiters/max-width`.
\end{itemize}
\begin{NiceMatrixBlock}{auto-columns-width}
\NiceMatrixOptions
\setlength{\extrarowheight}{1mm}
\begin{pNiceArray}{rrrr|r}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\
\end{pNiceArray}
\end{NiceMatrixBlock}

\begin{align*}
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\
\end{pmatrix} \\
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 & L_2 \gets L_1-4L_2 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1+4L_3 \\
0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1-4L_4 \\
\end{pmatrix} \\
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\
0 & 64 & -41 & 1 & 19 \\
\end{pmatrix} \\
\end{align*}

If you wish an alignment of the different matrices without the same width for all the columns, you can construct a unique array and place the parenthesis with commands \texttt{\SubMatrix} in the \texttt{\CodeAfter}. Of course, that array can’t be broken by a page break.

\begin{align*}
\begin{NiceMatrix}[ r, last-col=6, code-for-last-col = \scriptstyle \color{blue} ]
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1+4L_3 \\
0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1-4L_4 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
\CodeAfter [sub-matrix/vlines=4] \\
\SubMatrix{{1-1}{4-5}}
\end{NiceMatrix}
\end{align*}
16.5 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it’s possible to “draw” that cell with the key \texttt{draw} of the command \texttt{Block} (this is one of the uses of a mono-cell block\textsuperscript{42}).

\begin{pNiceArray}{>{\strut}cccc}
\textcolor{blue}{\Block}{}{a_{11}} & a_{12} & a_{13} & a_{14} \\
a_{21} & \Block{}{a_{22}} & a_{23} & a_{24} \\
a_{31} & a_{32} & \Block{}{a_{33}} & a_{34} \\
a_{41} & a_{42} & a_{43} & \Block{}{a_{44}}
\end{pNiceArray}

We should remark that the rules we have drawn are drawn \textit{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 \texttt{\hline}, the specifier “|” and the options \texttt{hlines}, \texttt{vlines} and \texttt{hvlines} spread the cells.\textsuperscript{43}

It’s possible to color a row with \texttt{\rowcolor} in the \texttt{\textbf{code-before}} (or with \texttt{\rowcolor} in the first cell of the row if the key \texttt{colortbl-like} is used—even when \texttt{colortbl} is not loaded).

\begin{pNiceArray}{>{\strut}cccc}
\rowcolor{red!15}A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & \rowcolor{red!15}A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & \rowcolor{red!15}A_{33} & A_{34} \\
A_{41} & A_{42} & A_{43} & \rowcolor{red!15}A_{44}
\end{pNiceArray}

\textsuperscript{42}We recall that, if the first mandatory argument of the command \texttt{Block} is left empty, that means that the block is a mono-cell block

\textsuperscript{43}For the command \texttt{\textbackslash cline}, see the remark p. 8.
However, it’s not possible to do a fine tuning. That’s why we describe now a 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.

**Caution:** Some PDF readers are not able to show transparency.

That example and the following ones require Tikz (by default, nicematrix only loads PGF, which is a sub-layer of Tikz) and the Tikz library fit. The following lines in the preamble of your document do the job:

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

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

\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}

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.

\textsuperscript{44}In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.
If we want to highlight each row of this matrix, we can use the previous technique three times.

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

16.6 Utilisation of \SubMatrix in the code-before

In the following example, we illustrate the mathematical product of two matrices. The whole figure is an environment \NiceArray and the three pairs of parenthesis have been added with \SubMatrix in the code-before. You will find the LaTeX code of that figure in the source file of this document.
17 Implementation

By default, the package nicematrix doesn’t patch any existing code. However, when the option `renew-dots` is used, the commands \cdots, \ldots, \vdots, 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. For example, 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

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

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

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

\begin{verbatim}
3 \RequirePackage{13keys2e}
4 \ProvidesExplPackage
5 \nicematrix
6 \myfiledate
7 \myfileversion
8 \Enhanced arrays with the help of PGF/TikZ
\end{verbatim}

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

We load some packages. The package xparse is still loaded for use on Overleaf.

\begin{verbatim}
9 \RequirePackage { xparse }
10 \RequirePackage { array }
11 \RequirePackage { amsmath }
\end{verbatim}
Technical definitions


In some constructions, we will have to use a \{pgfpicture\} which must be replaced by a \{tikzpicture\} if Tikz is loaded. However, this switch between \{pgfpicture\} and \{tikzpicture\} can’t be done dynamically with a conditional because, when the Tikz library external is loaded by the user, the pair \texttt{\begin{tikzpicture}-\end{tikzpicture}} (or \texttt{\begin{pgfpicture}-\end{pgfpicture}}) must be statically “visible” (even when externalization is not activated). That’s why we create \c_@@_pgfortikzpicture_tl and \c_@@_endpgfortikzpicture_tl which will be used to construct in a \AtBeginDocument the correct version of some commands.

We test whether the current class is \texttt{revtex4-1} (deprecated) or \texttt{revtex4-2} because these classes redefines \texttt{array} (of \texttt{array}) in a way incompatible with our programmation. At the date January 2021, the current version \texttt{revtex4-2} is 4.2e (compatible with \texttt{booktabs}).
Maybe one of the previous classes will be loaded inside another class... We try to detect that situation.

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

\ProvideDocumentCommand \iddots { }
\begin{verbatim}
\mathinner{
\tex_mkern:D 1 mu
\box_move_up:nn { 1 pt } \{ \hbox:n \{ . \} \}
\tex_mkern:D 2 mu
\box_move_up:nn { 4 pt } \{ \hbox:n \{ . \} \}
\tex_mkern:D 2 mu
\box_move_up:nn { 7 pt }
\{ \vbox:n \{ \kern 7 pt \hbox:n \{ . \} \} \}
\tex_mkern:D 1 mu
}
\end{verbatim}

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

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

\AtBeginDocument{
\@ifpackageloaded { booktabs }
{ \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
}
\cs_set_protected:Npn \nicematrix@redefine@check@rerun
{ \@@_old_pgfutil@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 created by nicematrix).
\begin{verbatim}
\cs_set_protected:Npn \pgfutil@check@rerun \nnn \@old_pgfutil@check@rerun \pgfutil@check@rerun
\end{verbatim}

We have to know whether colortbl is loaded in particular for the redefinition of \everycr.
\begin{verbatim}
\bool_new:N \c_@@_colortbl_loaded_bool
\AtBeginDocument{
\@ifpackageloaded { colortbl }
{ \bool_set_true:N \c_@@_colortbl_loaded_bool }
}
\end{verbatim}

The command \CT@arc@ is a command of colortbl which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if colortbl is not loaded.
\begin{verbatim}
\cs_set_protected:Npn \CT@arc@ { }
\cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 \#1 \#2 }
\{ \str_if_eq:eeF { mm- } \{ \tl_range:nnn \{ \#1 \} 1 3 \}
\{ \@@_old_pgfutil@check@rerun \#1 \#2 \} \}
}
\end{verbatim}

49
Idem for \CT@drs@.

\cs_set_protected:Npn \CT@drsc@ { }
\cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
\cs_set:Npn \CT@drs #1 #2 {
  \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
  { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
}
\cs_set:Npn \hline {
  \noalign { \ifnum 0 = `} \fi
  \cs_set_eq:NN \hskip \vskip
  \cs_set_eq:NN \vrule \hrule
  \cs_set_eq:NN \@width \@height
  { \CT@arc@ \vline }
  \futurelet \reserved@a \@xhline
}

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

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

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

\everycr { }
\cr
\noalign { \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 {
  \@xcline \l_@@_first_col_int }

We have to act in a fully expandable way since there may be \noalign (in the \multispan) to detect. That’s why we use \@@_cline_i:en.

\cs_set:Npn \@@_cline_i:en \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 {
  \CT@arc@ \leaders \hrule \@height \arrayrulewidth
}

Now, #1 is the number of the current column and we have to draw a line from the column #2 to the column #3 (both included).

\textsuperscript{45}See question 99041 on TeX StackExchange.
You look whether there is another \texttt{\textbackslash cline} to draw (the final user may put several \texttt{\textbackslash cline}).

The following commands are only for efficiency. They must \textit{not} be protected because it will be used (for instance) in names of \texttt{pgf} nodes.

The following command is a small shortcut.

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}.

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:

We want to patch this command (in the environments of \texttt{nicematrix}) in order to have:
However, we don’t want to use explicitly any private command of \textit{siunitx}. That’s why we will extract the name of the two \_\_\_\textit{siunitx}... commands by their position in the code of \texttt{\NC\rewrite@S}. Since the command \texttt{\NC\rewrite@S} appends some tokens to the \texttt{toks} list \texttt{\@temptokena}, we use the \LaTeX command \texttt{\NC\rewrite@S} in a group (\texttt{\groupbegin:\texttt{\groupend}}) and we extract the two command names which are in the toks \texttt{\@temptokena}. However, this extraction can be done only when \textit{siunitx} is loaded (and it may be loaded after \textit{nicematrix}) and, in fact, after the beginning of the document — because some instructions of \textit{siunitx} are executed in a \texttt{\AtBeginDocument}. That’s why this extraction will be done only at the first use of an environment of \textit{nicematrix} with the command \texttt{\@@\_adapt\_S\_column}.

\begin{verbatim}
\cs_set_protected:Npn \@@_adapt_S_column:
\end{verbatim}

The command \texttt{\@@\_renew\_NC\rewrite@S}: will be used in each environment of \textit{nicematrix} in order to “rewrite” the \texttt{S} column in each environment.
\@true_c: will be replaced statically by c at the end of the construction of the preamble.

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

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

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

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

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.

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

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

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

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

\begin{verbatim}
\tl_new:N \l_@@_cell_type_tl
\tl_set:Nn \l_@@_cell_type_tl { c }
\end{verbatim}

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

\begin{verbatim}
\dim_new:N \g_@@_blocks_wd_dim
\end{verbatim}

Idem pour the mono-row blocks.

\begin{verbatim}
\dim_new:N \g_@@_blocks_ht_dim
\dim_new:N \g_@@_blocks_dp_dim
\end{verbatim}

The sequence \texttt{\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 whether 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 \texttt{\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 boolean will be raised when the command \texttt{\rotate} is used.

\begin{verbatim}
\bool_new:N \g_@@_rotate_bool
\end{verbatim}
\cs_new_protected:Npn \@@_test_if_math_mode:
{
  \if_mode_math: \else:
    \@@_fatal:n { Outside~math~mode }
  \fi:
}

The letter used for the vlines which will be drawn only in the sub-matrices. \texttt{vlism} stands for \emph{vertical lines in sub-matrices}.
\tl_new:N \l_@@_letter_vlism_tl

The list of the columns where vertical lines in sub-matrices (vlism) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.
\seq_new:N \g_@@_cols_vlism_seq

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.
\colorlet { nicematrix-last-col } { . }
\colorlet { nicematrix-last-row } { . }

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

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}.
\tl_set:Nn \g_@@_com_or_env_str { environment }

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains \texttt{env}). This command must \texttt{not} be protected since it will be used in error messages and we have to use \texttt{\str_if_eq:VnTF} and not \texttt{\tl_if_eq:NnTF} because we need to be fully expandable).
\cs_new: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 \} }
}

The following token list corresponds to the option \texttt{code-after} (it’s also possible to set the value of that parameter with the keyword \texttt{\CodeAfter}).
\tl_new:N \g_nicematrix_code_after_tl

For the key \texttt{code} of the command \texttt{\SubMatrix} (itself in the main \texttt{\CodeAfter}), we will use the following token list.
\tl_new:N \l_@@_code_tl

The following token list has a function similar to \texttt{\g_nicematrix_code_after_tl} but it is used internally by \texttt{nicematrix}. In fact, we have to distinguish between \texttt{\g_nicematrix_code_after_tl} and \texttt{\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 \texttt{\l_@@_old_iRow_int} and \texttt{\l_@@_old_jCol_int} will be used to save the values of the potential \LaTeX counters \texttt{iRow} and \texttt{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 \texttt{rules/color} available in the environments.

\begin{verbatim}
\tl_new:N \l_@@_rules_color_tl
\bool_new:N \g_@@_after_col_zero_bool
\end{verbatim}

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if \texttt{\Hdotsfor} is used in that column.

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

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

It’s possible to use the command \texttt{\NotEmpty} to specify explicitly that a cell must be considered as non empty by \texttt{nicematrix} (the Tikz nodes are constructed only in the non empty cells).

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

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

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

- The final user can explicitly add material in \texttt{\l_@@_code_before_tl} by using the key \texttt{code-before} or the keyword \texttt{\CodeBefore} (with the keyword \texttt{\Body}).

\begin{verbatim}
\tl_new:N \l_@@_code_before_tl
\bool_new:N \l_@@_code_before_bool
\end{verbatim}

The following dimensions will be used when drawing the dotted lines.

\begin{verbatim}
\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
\end{verbatim}

\texttt{expl3} provides scratch dimensions \texttt{\l_tmpa_dim} and \texttt{\l_tmpb_dim}. We creates two more in the same spirit (if they don’t exist yet: that’s why we use \texttt{\dim_zero_new:N}).

\begin{verbatim}
\dim_zero_new:N \l_tmpc_dim
\dim_zero_new:N \l_tmpd_dim
\end{verbatim}

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

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

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

\begin{verbatim}
\dim_new:N \g_@@_width_last_col_dim
\dim_new:N \g_@@_width_first_col_dim
\end{verbatim}

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

\begin{verbatim}
\seq_new:N \g_@@_blocks_seq
\end{verbatim}
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).

The final user may decide to “stroke” a block (using, for example, the key \draw=red!15 when using the command \Block). In that case, the rules specified, for instance, by hvlines must not be drawn around the block. That’s why we keep the information of all that stroken blocks in the following sequence.

\seq_new:N \g_@@_pos_of_stroken_blocks_seq

The list of the names of the potential \SubMatrix in the \CodeAfter of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given \SubMatrix).

\seq_new:N \g_@@_submatrix_names_seq

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the \SubMatrix—the \SubMatrix in the code-before).

\int_new:N \l_@@_row_min_int
\int_new:N \l_@@_row_max_int
\int_new:N \l_@@_col_min_int
\int_new:N \l_@@_col_max_int

The following sequence will be used when the command \SubMatrix is used in the code-before (and not in the \CodeAfter). It will contain the position of all the sub-matrices specified in the code-before. Each sub-matrix is represented by an “object” of the forme \(\{i\}\{j\}\{k\}\{l\}\) where \(i\) and \(j\) are the number of row and column of the upper-left cell and \(k\) and \(l\) the number of row and column of the lower-right cell.

\seq_new:N \g_@@_submatrix_seq

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 \l_@@_static_num_of_col_int

The following parameters correspond to the keys fill, draw, borders and rounded-corners of the command \Block.

\tl_new:N \l_@@_fill_tl
\tl_new:N \l_@@_draw_tl
\clist_new:N \l_@@_borders_clist
\dim_new:N \l_@@_rounded_corners_dim

The following token list correspond to the key color of the command \Block. However, as of now (v. 5.7 of nicematrix), the key color linked to fill with an error. We will give to the key color of \Block its new meaning in a few months (with its new definition, the key color will draw the frame with the given color but also color the content of the block (that is to say the text) as does the key color of a Tikz node).

\tl_new:N \l_@@_color_tl
Here is the dimension for the width of the rule when a block (created by \Block) is stroked.

\dim_new:N \l_@@_line_width_dim

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

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

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

\bool_new:N \l_@@_draw_first_bool

The blocks which use the key – will store their content in a box. These boxes are numbered with the following counter.

\int_new:N \g_@@_block_box_int

\dim_new:N \l_@@_submatrix_extra_height_dim
\dim_new:N \l_@@_submatrix_left_xshift_dim
\dim_new:N \l_@@_submatrix_right_xshift_dim
\clist_new:N \l_@@_hlines_clist
\clist_new:N \l_@@_vlines_clist
\clist_new:N \l_@@_submatrix_hlines_clist
\clist_new:N \l_@@_submatrix_vlines_clist

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).

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

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”.

\bool_new:N \l_@@_last_row_without_value_bool

\footnote{We can’t use \l_@@_last_row_int for this usage because, if nicematrix has read its value from the aux file, the value of the counter won’t be −1 any longer.}
• 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. \texttt{\{bNiceMatrix\}}) and there is a last column but we don’t know its value because the user has used the option \texttt{last-col} without value. A value of 0 means that the option \texttt{last-col} has been used in an environment with preamble (like \texttt{\{pNiceArray\}}): in this case, the key was necessary without argument.

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

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

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}

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

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

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

\begin{verbatim}
\tl_new:N \l_@@_tabularnote_tl
\end{verbatim}

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

\begin{verbatim}
\int_new:N \l_@@_number_of_notes_int
\end{verbatim}

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

\begin{verbatim}
\cs_new:Npn \@@_notes_style:n #1 { \textit { \alph { #1 } } }
\end{verbatim}

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

\begin{verbatim}
\cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
\end{verbatim}
The following function can be redefined by using the key `notes/label-in-list`.

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

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

```latex\cs_set:Npn \thetabularnote { \@@_notes_style:n \{ tabularnote \} }```

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

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

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

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

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

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

```latex\NewDocumentCommand \tabularnote { m }
\bool_if:nTF { ! \c_@@_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. \textsuperscript{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.

367 \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
368 \peek_meaning:NF \tabularnote
369 {
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.

370 \hbox_set:Nn \l_tmpa_box
371 {
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
374 \stepcounter{tabularnote}
375 \@@_notes_style:n { \tabularnote }
376 \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
377 {
378 \stepcounter{tabularnote}
379 \@@_notes_style:n { \tabularnote }
380 }
381 \int_zero:N \l_@@_number_of_notes_int
382 \hbox_overlap_right:n { \box_use:N \l_tmpa_box }
383 We use \textsuperscript{a} 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 \textsuperscript{unskip} (inserted by \textsuperscript{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 }
\addtocounter{tabularnote}{1}

Command for creation of rectangle nodes

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

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

\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
394 \begin {pgfscope}
395 \pgfset
396 \outer-sep = \c_zero_dim ,
397 \inner-sep = \c_zero_dim ,
400 \end {pgfscope}
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}{\pgfpoint{#2}{#3}}}
\pgfgetlastxy\l_tmpa_dim\l_tmpb_dim
\pgfnode{rectangle}{center}{
vbox_to_ht:nn{\dim_abs:n\l_tmpb_dim}
vfill\hbox_to_wd:nn{\dim_abs:n\l_tmpa_dim}{}}
\end{pgfscope}
\end{verbatim}

The options

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

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

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

\begin{verbatim}\bool_new:N \l_@@_standard_cline_bool\end{verbatim}
The following dimensions correspond to the options `cell-space-top-limit` and co (these parameters are inspired by the package `cellspace`).

\[ \dim_new:N \l_@@_cell_space_top_limit_dim \]
\[ \dim_new:N \l_@@_cell_space_bottom_limit_dim \]

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.

\[ \dim_new:N \l_@@_inter_dots_dim \]
\[ \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 \text{ em} } } \]

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even though it is obsolete).

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

\[ \dim_new:N \l_@@_xdots_shorten_dim \]
\[ \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 \text{ em} } } \]

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even though it is obsolete).

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

\[ \dim_new:N \l_@@_radius_dim \]
\[ \AtBeginDocument { \dim_set:Nn \l_@@_radius_dim { 0.53 \text{ pt} } } \]

The `\AtBeginDocument` is only a security in case `revtex4-1` is used (even if it is obsolete).

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

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

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

\[ \bool_new:N \l_@@_light_syntax_bool \]

The string `\l_@@_baseline_tl` 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).

\[ \tl_new:N \l_@@_baseline_tl \]
\[ \tl_set:Nn \l_@@_baseline_tl { c } \]

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

\[ \bool_new:N \l_@@_exterior_arraycolsep_bool \]

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

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

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 `hvlines-except-corners` raises the key `except-corners`.

\[ \clist_new:N \l_@@_except_corners_clist \]
\newdimen\l_@@_notes_above_space_dim
\AtBeginDocument{\dim_set:Nn \l_@@_notes_above_space_dim {1 \text{ mm}}} 

The \AtBeginDocument is only a security in case revtex4-1 is used (even if it is obsolete).

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 but not the dotted lines drawn by \hdottedline and “:”.

\tl_new:N \l_@@_xdots_color_tl

The following token list corresponds to the key delimiters/color.

\tl_new:N \l_@@_delimiters_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 delimiters/max-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_@@_delimiters_max_width_bool
We can’t use \c_@@_tikz_loaded_bool to test whether tikz is loaded because \NiceMatrixOptions may be used in the preamble of the document.

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

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

First, we define a set of keys “NiceMatrix / Global” which will be used (with the mechanism of \inherit:n) by other sets of keys.
With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

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

```latex
\keys_define:nn { NiceMatrix / Env }
{ delimiters/max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
  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 } }
}\endinput
```
The options `c`, `t`, and `b` of the environment `{\texttt{NiceArray}}` have the same meaning as the option of the classical environment `{\texttt{array}}).

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.
We begin the construction of the major sets of keys (used by the different user commands and environments).

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

\keys_define:nn { NiceMatrix / NiceMatrixOptions }
{
  delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
  delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
  delimiters / color .value_required:n = true ,
  delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
  delimiters-color .value_required:n = true ,
  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 ,
With the option renew-matrix, the environment \{matrix\} of amsmath and its variants are redefined to behave like the environment \{NiceMatrix\} and its variants.

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

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

\begin{verbatim}
transparent .code:n =
  \{ \@@_renew_matrix: ,
    \bool_set_true:N \l_@@_renew_dots_bool
  \} ,
transparent .value_forbidden:n = true ,
\end{verbatim}

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

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

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

In \NiceMatrixOptions, the special value auto is not available.

\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 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}

69
By default, the specifier used in the preamble of the array (for example in \{pNiceArray\}) to draw a vertical dotted line between two columns is the colon ‘:\’. However, it’s possible to change this letter with \texttt{letter-for-dotted-lines} and, by the way, the letter ‘:\’ will remain free for other packages (for example alyshah).

\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 { One-letter-allowed } }
},
letter-for-dotted-lines .value_required:n = true ,
notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
notes .value_required:n = true ,
sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
sub-matrix .value_required:n = true ,
unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
}
\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.

\begin{Verbatim}
\texttt{\textbackslash NewDocumentCommand \textbackslash NiceMatrixOptions \{ m \}}
{ \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
\end{Verbatim}

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

\begin{Verbatim}
\keys_define:nn { NiceMatrix / NiceMatrix }
{ 
last-col .code:n = \tl_if_empty:nTF {#1} 
{ 
\bool_set_true:N \l_@@_last_col_without_value_bool
\int_set:Nn \l_@@_last_col_int { -1 }
} ,
{ \int_set:Nn \l_@@_last_col_int { #1 } } ,
l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
small .bool_set:N = \l_@@_small_bool ,
small .value_forbidden:n = true ,
delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
delimiters / color .value_required:n = true ,
delimiters-color .tl_set:N = \l_@@_delimiters_color_tl ,
delimiters-color .value_required:n = true ,
unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix } }
\end{Verbatim}

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

\begin{Verbatim}
\keys_define:nn { NiceMatrix / 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.

\begin{Verbatim}
small .bool_set:N = \l_@@_small_bool ,
small .value_forbidden:n = true ,
last-col .code:n = \tl_if_empty:nF { #1 }
{ \@@_error:n { last-col-non-empty-for-NiceArray } }
\int_zero:N \l_@@_last_col_int ,
notes / para .bool_set:N = \l_@@_notes_para_bool ,
\end{Verbatim}

70
We finalise the definition of the set of keys “NiceMatrix / NiceTabular” with the options specific to \{NiceTabular\}.

Important code used by \{NiceArrayWithDelims\}

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

At the beginning of the cell, we link \CodeAfter to a command which do not begin with \omit (whereas the standard version of \CodeAfter begins with \omit).

We increment \c@jCol, which is the counter of the columns.
Now, we increment the counter of the rows. We don’t do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don’t want to take into account.

```latex
\int_compare:nNnT \c@jCol = 1
\{ \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: \}
\int_gset:Nn \g_@@_col_total_int \{ \int_max:nn \g_@@_col_total_int \c@jCol \}
```

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_end:` corresponding to this `\bbox_set:Nw` will be in the \@@_end_Cell: (and the potential `\c_math_toggle_token` also).

```latex
\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.

```latex
\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.

```latex
\cs_new_protected:Npn \@@_begin_of_row:
{ \int_gincr:N \c@iRow
\dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
\dim_gset:Nn \g_@@_dp_last_row_dim \{ \box_dp:N \arstrutbox \}
\dim_gset:Nn \g_@@_ht_last_row_dim \{ \box_ht:N \arstrutbox \}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pdfcoordinate
\pgfreememberpicturepositiononpagetrue
\pdfcoordinate
{ \@@_env: - row - \int_use:N \c@iRow - base }
{ \pgfpoint \c_zero_dim \{ 0.5 \arrayrulewidth \} }
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias
{ \l_@@_name_str - row - \int_use:N \c@iRow - base }
{ \@@_env: - row - \int_use:N \c@iRow - base }
}
\endpgfpicture
```

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.
\cs_new_protected:Npn \@@_update_for_first_and_last_row:
\{ 
\int_compare:nNnT \c@iRow = 0
\{ \dim_gset:Nn \g_@@_dp_row_zero_dim
\} \dim_gset:Nn \g_@@_ht_row_zero_dim
\{ \dim_max:nn \g_@@_dp_row_zero_dim \} \{ \box_dp:N \l_@@_cell_box \}
\dim_gset:Nn \g_@@_ht_row_zero_dim
\{ \dim_max:nn \g_@@_ht_row_zero_dim \} \{ \box_ht:N \l_@@_cell_box \}
\}
\int_compare:nNnT \c@iRow = 1
\{ \dim_gset:Nn \g_@@_ht_row_one_dim
\} \dim_gset:Nn \g_@@_ht_row_one_dim
\{ \dim_max:nn \g_@@_ht_row_one_dim \} \{ \box_ht:N \l_@@_cell_box \}
\}
\cs_new_protected:Npn \@@_rotate_cell_box:
\{ \box_rotate:Nn \l_@@_cell_box { 90 }
\int_compare:nNnT \c@iRow = \l_@@_last_row_int
\{ \vbox_set_top:Nn \l_@@_cell_box
\} \dim_gset:Nn \g_@@_ht_row_zero_dim
\{ \dim_max:nn \g_@@_ht_row_zero_dim \} \{ \box_ht:N \l_@@_cell_box
\}
\bool_gset_false:N \g_@@_rotate_bool
\}
\cs_new_protected:Npn \@@_adjust_size_box:
\{ \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
\{ \box_set_wd:Nn \l_@@_cell_box
\} \dim_gzero:N \g_@@_blocks_wd_dim
\}
\dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
\{ \box_set_dp:Nn \l_@@_cell_box
\} \dim_gzero:N \g_@@_blocks_dp_dim
\}
\dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
\{ \box_set_ht:Nn \l_@@_cell_box
\} \dim_gzero:N \g_@@_blocks_ht_dim
\}
\cs_new_protected:Npn \@@_end_Cell:
\{ \@@_math_toggle_token:
\hbox_set_end:
\bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
\@@_adjust_size_box:
\box_set_ht:Nn \l_@@_cell_box
\{ \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim \}
\box_set_dp:Nn \l_@@_cell_box
\{ \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim \}
We want to compute in \texttt{\_\_\_max\_cell\_width\_dim} the width of the widest cell of the array (except the cells of the "first column" and the "last column").

\begin{verbatim}
\dim_gset:Nn \_\_\_max\_cell\_width\_dim
\{ \dim_max:nn \_\_\_max\_cell\_width\_dim \{ \box_wd:N \l_@@_cell_box \} \}
\end{verbatim}

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 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 for now, we use the following technic:

- if the width of the box \texttt{\_\_\_cell\_box} (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a \texttt{\rlap}, a \texttt{\llap} or a \texttt{\mathclap} of \texttt{mathtools}.

- the cells with a command \texttt{\Ldots} or \texttt{\Cdots}, \texttt{\Vdots}, etc., should also be considered as empty: if \texttt{nullify-dots} is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of \texttt{CodeAfter}); however, if \texttt{nullify-dots} is not in force, a phantom of \texttt{\ldots}, \texttt{\cdots}, \texttt{\vdots} is inserted and its width is not equal to zero; that’s why these commands raise a boolean \texttt{\_\_\_empty\_cell\_bool} and we begin by testing this boolean.

\begin{verbatim}
\bool_if:NTF \_\_\_empty\_cell\_bool
{ \box_use_drop:N \_\_\_cell\_box }
{ \bool_lazy_or:nnTF
\_\_\_not\_empty\_cell\_bool
{ \dim_compare_p:nNn { \box_wd:N \_\_\_cell\_box } > \c_zero_dim }
\_\_\_node\_for\_the\_cell:
{ \box_use_drop:N \_\_\_cell\_box }
}
\bool_gset_rTF \_\_\_empty\_cell\_bool
\_\_\_not\_empty\_cell\_bool
\end{verbatim}

The following command creates the PGF name of the node with, of course, \texttt{\_\_\_cell\_box} as the content.

\begin{verbatim}
\cs_new_protected:Npn \_\_\_node\_for\_the\_cell:
{ \pgfpicture
\pgfsetbaseline \c_zero_dim
\pgfrememberpicturepositionnonpagetrue
\pgfset
{ inner-sep = \c_zero_dim ,
  minimum-width = \c_zero_dim }
\pgfnode
{ rectangle }
{ base }
{ \box_use_drop:N \_\_\_cell\_box }
{ \_\_\_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
{ }
\str_if_empty:NT \_\_\_name\_str
{ \pgfnodealias
{ \_\_\_name\_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
{ \_\_\_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
\end{verbatim}
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 \g_@@_type_lines_tl the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,
\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\ 5 & \text{Cdots} & & 6 \\ 7 & \text{Cdots}[\text{color=red}] \\
\end{pNiceMatrix}
the content of \g_@@_Cdots_lines_tl will be:
\@@_draw_Cdots:nnn {2}{2}{1}
\@@_draw_Cdots:nnn {3}{2}{\text{color=red}}

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.

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.

It colortbl is loaded, \@tabarray has been redefined to incorporate \CT@start.
\_@@_baseline_tl 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. Remark that \str_if_eq:VnTF is fully expandable and you need something fully expandable here.

\begin{verbatim}
[ \str_if_eq:VnTF \_@@_baseline_tl c c t ]
\end{verbatim}

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

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

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

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

The \hbox:n (or \hbox) is mandatory.

\begin{verbatim}
\hbox
\begin{verbatim}
\bool_if:NT \_@@_code_before_bool
  \begin{verbatim}
  \vtop
  \skip_vertical:N \_0.5\arrayrulewidth
  \pgfsys@markposition { \@@_env: - row - \_@@_succ:n \_0\_iRow }
  \skip_vertical:N -\_0.5\arrayrulewidth
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}

The following must not be protected because it begins with \noalign.

\begin{verbatim}
\cs_new:Npn \_@@_everycr: { \noalign { \_@@_everycr_i: } }
\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}
\tl_if_empty:NF \_\_0\_hlines_clist
\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}
\int_compare:nNnT \c@iRow > \{-1\} \\
\int_compare:nNnF \c@iRow = \l_@@_last_row_int
\end{verbatim}

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

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

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

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

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

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

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

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

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

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_array_ii: \\
\IfBoolexpr: X { \booktabs_loaded \detect_verticalskip + \usebox { \@l_\texttt{detect vertical place} } \\ \\
\IfBoolexpr: X { \booktabs_loaded \detect_verticalskip - \usebox { \@l_\texttt{detect vertical place} } }
\end{verbatim}

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

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

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

\footnote{The option `small` of \nicematrix changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.}
We keep in memory the old versions or \ldots, \cdots, etc. only because we use them inside \phantom commands in order that the new commands \Ldots, \Cdots, etc. give the same spacing (except when the option nullify-dots is used).

\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 \Hdotsfor \@@_Hdotsfor:
\cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
\cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
\cs_set_eq:NN \Block \@@_Block:
\cs_set_eq:NN \rotate \@@_rotate:
\cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix
\cs_set_eq:NN \dotfill \@@_old_dotfill:
\cs_set_eq:NN \CodeAfter \@@_CodeAfter:
\cs_set_eq:NN \NotEmpty \@@_NotEmpty:
\bool_if:NT \l_@@_standard_cline_bool \@@_standard_cline:
\bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
\seq_gclear_new:N \g_@@_multicolumn_cells_seq
\seq_gclear_new:N \g_@@_multicolumn_sizes_seq
\int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }
\int_gzero_new:N \g_@@_row_total_int
\int_gzero_new:N \g_@@_col_total_int
\bool_gset_false:N \g_@@_last_col_found_bool
\tl_gclear_new:N \g_@@_Cdots_lines_tl
\tl_gclear_new:N \g_@@_Ldots_lines_tl
\tl_gclear_new:N \g_@@_Vdots_lines_tl
\tl_gclear_new:N \g_@@_Ddots_lines_tl
\seq_gclear_new:N \g_@@_multicolumn_cells_seq
\seq_gclear_new:N \g_@@_multicolumn_sizes_seq
\end{verbatim}

The sequence $\g_@@_multicolumn_cells_seq$ will contain the list of the cells of the array where a command $\multicolumn{n}{...}{...}$ with $n > 1$ is issued. In $\g_@@_multicolumn_sizes_seq$, the “sizes” (that is to say the values of $n$) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

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

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 \CodeAfter) they represent the numbers of rows and columns of the array (without the potential last row and last column).

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

Now, we will create all the col nodes and row nodes with the informations written in the aux file. You use the technique described in the page 1229 of pgfmanual.pdf, version 3.1.4b.
Now, you create the diagonal nodes by using the \texttt{row} nodes and the \texttt{col} nodes. If the engine is $\texttt{xetex}$ or $\texttt{luatex}$ we also create the “½ nodes”.

\@@_create_diag_nodes:

Now, yet other settings before the execution of the \texttt{code-before}.

\begin{verbatim}
\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 \roundedrectanglecolor \@@_roundedrectanglecolor
\cs_set_eq:NN \rowcolor \@@_rowcolor
\cs_set_eq:NN \rowcolors \@@_rowcolors
\cs_set_eq:NN \arraycolor \@@_arraycolor
\cs_set_eq:NN \columncolor \@@_columncolor
\cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
\cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
\end{verbatim}

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}.

\begin{verbatim}
\bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
\exp_last_unbraced:NV \@@_CodeAfter_keys: \l_@@_code_before_tl
\end{verbatim}

Here is the \texttt{CodeBefore}. As of now, the keys that may be provided to the keyword \texttt{CodeBefore} are the same as keys that may be provided to \texttt{CodeAfter}, hence the \texttt{@@_CodeAfter_keys}:

\begin{verbatim}
\@\_actually_color:
\bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
\group_end:
\end{verbatim}

A value of $-1$ for the counter $\lfloor 1_\@_last\_row\rfloor$ 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
{ \}
\}
\}
\end{verbatim}

A value based on the name is more reliable than a value based on the number of the environment.
A value of \(-1\) for the counter \(\l_\@@_last_col_int\) means that the user has used the option \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 \texttt{aux} file (if it has been written on a previous run).

```latex
\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_\@@_name_str } \{
\int_set:Nn \l_\@@_last_col_int \{ \use:c { \@@_last_col_ \l_\@@_name_str } \}
\}
\}
\}
```

The code in \texttt{\@@_pre_array_ii} is used only by \texttt{\textit{NiceArrayWithDelims}}.

The command \texttt{\bBigg@} is a command of \texttt{amsmath}.

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

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

Now, the preamble is constructed in \g_@@_preamble_tl.

Here is the beginning of the box which will contain the array. The \hbox_set_end: corresponding to this \hbox_set:Nw will be in the second part of the environment (and the closing \c_math_toggle_token also).

\hbox_set:Nw \l_@@_the_array_box
\skip_horizontal:N \l_@@_left_margin_dim
\skip_horizontal:N \l_@@_extra_left_margin_dim
\c_math_toggle_token
\bool_if:NTF \l_@@_light_syntax_bool
{ \use:c { @@-light-syntax }
{ \use:c { @@-normal-syntax }
}
\hbox_set:Nw \l_@@_code_before_tl { \CodeBefore
\bool_set_true:N \l_@@_code_before_bool
\hbox_set:Nw \l_@@_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 }
}
\begin{NiceArrayWithDelims}
\NewDocumentEnvironment { NiceArrayWithDelims }
{ m m O { } m ! O { } t \CodeBefore }
\begin{NiceArrayWithDelims}

The aim of the following \bgroup (the corresponding \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.

\bgroup
\tl_set:Nn \l_@@_left_delim_tl { #1 }
\tl_set:Nn \l_@@_right_delim_tl { #2 }
\tl_gset:Nn \g_@@_preamble_tl { #4 }
\int_gzero:N \g_@@_block_box_int
\dim_zero:N \g_@@_width_last_col_dim
\dim_zero:N \g_@@_width_first_col_dim
\bool_gset_false:N \g_@@_row_of_col_done_bool
\str_if_empty:NT \g_@@_name_env_str
{ \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
\bgroup
\tl_set:Nn \l_@@_left_delim_tl { #1 }
\tl_set:Nn \l_@@_right_delim_tl { #2 }
\tl_gset:Nn \g_@@_preamble_tl { #4 }
\int_gzero:N \g_@@_block_box_int
\dim_zero:N \g_@@_width_last_col_dim
\dim_zero:N \g_@@_width_first_col_dim
\bool_gset_false:N \g_@@_row_of_col_done_bool
\str_if_empty:NT \g_@@_name_env_str
{ \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
\bgroup
\tl_set:Nn \l_@@_left_delim_tl { #1 }
\tl_set:Nn \l_@@_right_delim_tl { #2 }
\tl_gset:Nn \g_@@_preamble_tl { #4 }
\int_gzero:N \g_@@_block_box_int
\dim_zero:N \g_@@_width_last_col_dim
\dim_zero:N \g_@@_width_first_col_dim
\bool_gset_false:N \g_@@_row_of_col_done_bool
\str_if_empty:NT \g_@@_name_env_str
{ \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
\bgroup

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.

\cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

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

\cs_if_exist:NT \tikz@library@external@loaded {
\tikzexternaldisable
\cs_if_exist:NT \ifstandalone {
\tikzset{external/optimize=false}
}
}

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

\int_gincr:N \g_@@_env_int

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

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.

\bool_if:NTF \l_@@_nicearray_bool {
\keys_set:nn { NiceMatrix / NiceArray } 
\keys_set:nn { NiceMatrix / pNiceArray } 
#3 , #5 }

\tl_if_empty:NT \l_@@_rules_color_tl {
\exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }

The argument \#6 is the last argument of \{NiceArrayWithDelims\}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It’s the job that will do the command \@@_pre_array_i:w. After that job, the command \@@_pre_array_i:w will go on with \@@_pre_array:w.

\IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:w
{
\bool_if:NTF \l_@@_light_syntax_bool {
\use:c { end @@-light-syntax }
\use:c { end @@-normal-syntax }
}
\tl_if_empty:NT \l_@@_rules_color_tl {
\exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
\bigskip

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

\begin{verbatim}
\int_compare:nNnT \l_@@_last_row_int > \c@iRow
{ \bool_if:NF \l_@@_last_row_without_value_bool
{ \int_compare:nNnF \l_@@_last_row_int = \c@iRow
{ \@@_error:n { Wrong~last~row }
\int_gset_eq:NN \l_@@_last_row_int \c@iRow
}
}
\int_gset_eq:NN \c@jCol \g_@@_col_total_int
\bool_if:nTF \g_@@_last_col_found Bool
{ \int_gdecr:N \c@jCol }
{ \int_compare:nNnT \l_@@_last_col_int > \c@iRow
{ \@@_error:n { last~col~not~used }
}
}
\int_gset_eq:NN \g_@@_row_total_int \c@iRow
\int_compare:nNnT \l_@@_last_row_int > \c@iRow
{ \int_gdecr:N \c@iRow }
\end{verbatim}

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

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

Now, the definition of \c@jCol and \g_@@_col_total_int change: \c@jCol will be the number of columns without the “last column”; \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 }
{ \int_compare:nNnT \l_@@_last_col_int > \c@iRow
{ \@@_error:n { last~col~not~used }
}
}
\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 \g_@@_width_first_col_dim: see p. 102).

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

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

Remark that, in all cases, \@@_use_arraybox_with_notes_c: is used.

\begin{verbatim}
\bool_if:NTF \l_@@_NiceArray_bool
{ \str_case:VnF \l_@@_baseline_tl
{ 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_@@_first_row_int which is the total height of the “first row” above the array (when the key first-row is used).

\begin{verbatim}
\int_compare:nNnT \l_@@_first_row_int = 0
{ \int_compare:nNnT \l_@@_first_row_int = 0
}
\end{verbatim}

\footnote{We remind that the potential “first column” (exterior) has the number 0.}
We compute $\l_{\text{tmpb}}$ 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 $\l_{\text{last_row_int}}$ means that there is no “last row.”

\begin{verbatim}
\int_compare:nNnTF \l_{\text{last_row_int}} > { -2 } 
\{ 
\dim_set_eq:NN \l_{\text{tmpb}} \g_{\text{ht_last_row_dim}} 
\dim_add:Nn \l_{\text{tmpb}} \g_{\text{dp_last_row_dim}} 
\} 
\dim_zero:N \l_{\text{tmpb}} 
\hbox_set:Nn \l_{\text{tmpa_box}} 
\{ 
\c_math_toggle_token 
\tl_if_empty:NF \l_{\text{delimiters_color_tl}} 
\{ \color { \l_{\text{delimiters_color_tl}} } \} 
\exp_after:wN \left \l_{\text{left_delim_tl}} 
vcenter 
\skip_vertical:N -\l_{\text{tmpa_dim}} 
\hbox 
\{ 
\bool_if:NTF \l_{\text{Matrix_bool}} \g_{\text{col@sep}} 
\{ \skip_horizontal:N -\arraycolsep \} 
\bool_if:NT \g_{\text{last_col_found_bool}} 
\} 
\skip_horizontal:N \g_{\text{width_last_col_dim}} 
\skip_horizontal:N \col@sep 
\} 
\bool_if:NF \l_{\text{Matrix_bool}} 
\{ \skip_horizontal:N -\arraycolsep \} 
\} 
\} 
\end{verbatim}

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

\begin{verbatim}
\tl_if_empty:NF \l_{\text{delimiters_color_tl}} 
\{ \color { \l_{\text{delimiters_color_tl}} } \} 
\exp_after:wN \right \l_{\text{right_delim_tl}} 
\c_math_toggle_token 
\} 
\end{verbatim}

Now, the box $\l_{\text{tmpa_box}}$ is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option \texttt{delimiters/max-width} is used.

\begin{verbatim}
\bool_if:NTF \l_{\text{delimiters_max_width_bool}} 
\{ \@@_put_box_in_flow_bis:nn \{ \#1 \}, \{ \#2 \} \} 
\@@_put_box_in_flow: 
\end{verbatim}

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 $\g_{\text{width_last_col_dim}}$: see p. 103).

\begin{verbatim}
\bool_if:NT \g_{\text{last_col_found_bool}} 
\{ \skip_horizontal:N \g_{\text{width_last_col_dim}} 
\skip_horizontal:N \g_{\text{col@sep}} 
\} 
\bool_if:NT \l_{\text{Matrix_bool}} 
\end{verbatim}

\footnote{A value of $-1$ for $\l_{\text{last_row_int}}$ means that there is a “last row” but the user have not set the value with the option \texttt{last row} (and we are in the first compilation).}
The aim of the following \egroup (the corresponding \bgroup is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

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

We construct the preamble of the array

The preamble given by the final use is in \g_@@_preamble_tl and the modified version will be stored in \g_@@_preamble_tl also.

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 columns type expanded because we will do the patch ourselves after. We want to be able the standard column types \texttt{w} and \texttt{W} in potential \{tabular\} of \texttt{array} in some cells of our array. That’s why we do those redefinitions in a \texttt{TeX} 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).

First, we have to store our preamble in the token register \temptokena (those “token registers” are not supported by expl3).

Initialisation of a flag used by \texttt{array} to detect the end of the expansion.

The following line actually does the expansion (it’s has been copied from \texttt{array.sty}).

Now, we have to “patch” that preamble by transforming some columns. We will insert in the \texttt{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 \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}.
The sequence \texttt{\g@@cols_vlism_seq} will contain the numbers of the columns where you will have to draw vertical lines in the potential sub-matrices (hence the name \texttt{vlism}).

The counter \texttt{\l_tmpa_int} will count the number of consecutive occurrences of the symbol \texttt{|}.

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

Now, we replace \texttt{\columncolor} by \texttt{\@@\columncolor_preamble}.

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

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

88
We add a last column to raise a good error message when the user put more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in \{NiceTabular\} ($\backslash\_@@\_tabular\_width\_dim=0pt$).

\begin{verbatim}
\dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
{ \tl_gput_right:Nn \g_@@_preamble_tl
  { > { \@@_error_too_much_cols: } l }
}
\end{verbatim}

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

\begin{verbatim}
\group_end:
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble:n #1
{ \str_case:nnF { #1 } { c { \@@_patch_preamble_i:n #1 }
  l { \@@_patch_preamble_i:n #1 }
  r { \@@_patch_preamble_i:n #1 }
  > { \@@_patch_preamble_ii:nn #1 }
  ! { \@@_patch_preamble_ii:nn #1 }
  @ { \@@_patch_preamble_ii:nn #1 }
  | { \@@_patch_preamble_iii:n #1 }
  p { \@@_patch_preamble_iv:nnn t #1 }
  m { \@@_patch_preamble_iv:nnn c #1 }
  b { \@@_patch_preamble_iv:nnn b #1 }
  \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
  \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
  \@@_true_c: { \@@_patch_preamble_vi:n #1 }
  ( { \@@_patch_preamble_vii:n #1 }
  ) { \@@_patch_preamble_vii:n #1 }
  \} { \@@_patch_preamble_vii:n #1 }
  \} { \@@_patch_preamble_vii:n #1 }
  \} { \@@_patch_preamble_vii:n #1 }
  \) { \@@_patch_preamble_vii:n #1 }
  C { \@@_error:nn { old-column-type } #1 }
  L { \@@_error:nn { old-column-type } #1 }
  R { \@@_error:nn { old-column-type } #1 }
  \q_stop { }
}

\str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
{ \@@_patch_preamble_vi:n #1 }
\end{verbatim}
For \texttt{c, l and r}:
\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_i:n #1
\{\
 \tl_gput_right:Nn \g_@@_preamble_tl
 \{ > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } } \}
 #1
 < \@@_end_Cell:
\}
\end{verbatim}

We increment the counter of columns and then we test for the presence of a $<$.
\begin{verbatim}
\int_gincr:N \c@jCol
\@@_patch_preamble_x:n
\end{verbatim}

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

For $|$:
\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_iii:n #1
\{\
 \l_tmpa_int is the number of successive occurrences of $|$.
\int_incr:N \l_tmpa_int
\@@_patch_preamble_iii_i:n
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
\{\
 \str_if_eq:nnTF { #1 } |\
 \{ \@@_patch_preamble_iii_i:n | \}
 \{\
 \tl_gput_right:Nx \g_@@_preamble_tl
 \{\
 \exp_not:N !\
 \{\
 \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
\}
\end{verbatim}
For $p$, $m$ and $b$

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
\{
  \tl_gput_right:Nn \g_@@_preamble_tl
  \{
    > { \@@_Cell:
      \begin { minipage } [ #1 ] { #3 } \mode_leave_vertical:
        \arraybackslash
      \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt \% v. 5.11
    }
    c < { \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt \% v. 5.11
      \end { minipage }
    } \@@_end_Cell:
  }
\}
\end{verbatim}

We increment the counter of columns, and then we test for the presence of a $<$.  

\begin{verbatim}
\int_gincr:N \c@jCol
\@@_patch_preamble_x:n
\end{verbatim}

For $w$ and $W$

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
\{
  \tl_gput_right:Nn \g_@@_preamble_tl
  \{
    > { \hbox_set:Nw \l_@@_cell_box \@@_Cell:
      \tl_set:Nn \l_@@_cell_type_tl { #3 }
    }
    c < { \@@_end_Cell:
      #1 \hbox_set_end:
      \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
        \@@_adjust_size_box:
        \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
    }
  }
\}
\end{verbatim}

We increment the counter of columns and then we test for the presence of a $<$.  

\begin{verbatim}
\int_gincr:N \c@jCol
\@@_patch_preamble_x:n
\end{verbatim}

For $\@@_true_c$: which will appear in our redefinition of the columns of type $S$ (of siunitx).

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_vi:n #1
\{
  \tl_gput_right:Nn \g_@@_preamble_tl \{ c }
\}
\end{verbatim}

We increment the counter of columns and then we test for the presence of a $<$.  

\begin{verbatim}
\int_gincr:N \c@jCol
\@@_patch_preamble_x:n
\end{verbatim}

For $\$, [ and \{.

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_vii:n #1
\{
\end{verbatim}
If we are before the column 1, we reserve space for the left delimiter.

\int_compare:nNnT \c@jCol = \c_zero_int
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_true_bool
\@@_patch_preamble:n

After this closing delimiter, we have to test whether there is an opening delimiter (we consider that
there can’t be a second closing delimiter) in order to add an horizontal space (equal to 0.5 em).

\@@_patch_preamble_viii_i:n
\cs_new_protected:Npn \@@_patch_preamble_viii_i:n #1
\bool_lazy_any:nT
\str_if_eq:nnTF { #1 } { ( } \@@_patch_preamble_ix:n
\tl_if_eq:NnTF \l_@@_vlines_clist { all }
\exp_args:NNx \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
\@@_patch_preamble:n
\tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
\@@_patch_preamble:n
\cs_new_protected:Npn \@@_patch_preamble_x:n #1
\str_if_eq:nnTF { #1 } { < }
\@_patch_preamble_ix:n
\tl_gput_right:Nn \g_@@_preamble_tl
\tl_gput_right:Nn \g_@@_preamble_tl
\exp_args:NNx \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
\{ 

The command \@@_vdottedline:n is protected, and, therefore, won’t be expanded before writing on \g_@@_internal_code_after_tl.

\tl_gput_right:Nx \g_@@_internal_code_after_tl
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\@@_vdottedline:n \{ \int_use:N \c@jCol \}
\@@_patch_preamble:n
\tl_gput_right:Nx \g_@@_preamble_tl
\tl_gput_right:Nx \g_@@_preamble_tl
\tl_gput_right:Nx \g_@@_preamble_tl
\{ \int_use:N \arrayrulewidth \}
\exp_args:NNx \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
\{ 

After a specifier of column, we have to test whether there is one or several <{..} because, after those
typical <{..}, we have to insert !\{\skip_horizontal:N ...\} when the key \texttt{vlines} is used.

\cs_new_protected:Npn \@@_patch_preamble_x:n #1
\str_if_eq:nnTF { #1 } { < }
\@_patch_preamble_ix:n
\tl_gput_right:Nn \l_@@_vlines_clist { \{ all } 
\tl_gput_right:Nx \g_@@_preamble_tl
\{ \! \{ \skip_horizontal:N \arrayrulewidth \} 
\}
\}
\exp_args:NNx \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
\{ 

The command `\@@_put_box_in_flow:` puts the box `\l_tmpa_box` (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the depth to take back into account the potential exterior rows (the total height of the first row has been computed in `\l_tmpa_dim` and the total height of the potential last row in `\l_tmpb_dim`).

```latex
\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 }
\tl_if_eq:NnTF \l_@@_baseline_tl { c }{ \box_use_drop:N \l_tmpa_box }
\@@_put_box_in_flow_i:
```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different of `c` (which is the initial value and the most used).

```latex
\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_tl { line- }{ \int_set:Nn \l_tmpa_int \str_range:Nnn \l_@@_baseline_tl 6 \tl_count:V \l_@@_baseline_tl }
\@@_qpoint:n { row - \int_use:N \l_tmpa_int }
```

Now, `\g_tmpa_dim` contains the y-value of the center of the array (the delimiters are centered in relation with this value).

```latex
\str_if_in:NnTF \l_@@_baseline_tl { line- }{ \int_set:Nn \l_tmpa_int \str_range:Nnn \l_@@_baseline_tl 6 \tl_count:V \l_@@_baseline_tl }
\@@_qpoint:n { row - \int_use:N \l_tmpa_int }
```

Now, `\g_tmpa_dim` contains the y-value of the center of the array (the delimiters are centered in relation with this value).
\int_set:Nn \l_tmpa_int 1
\@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

We take into account the position of the mathematical axis.
\dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
\dim_gsub:Nn \g_tmpa_dim \pgf@y

Now, \g_tmpa_dim contains the value of the $y$ translation we have to do.
\endpgfpicture \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
\box_use_drop:N \l_tmpa_box

The following command is always used by \{NiceArrayWithDelims\} (even if, in fact, there is no tabular notes: in fact, it’s not possible to know whether there is tabular notes or not before the composition of the blocks).
\cs_new_protected:Npn \@@_use_arraybox_with_notes_c: {
  \begin { minipage } \[ t \] { \boxed { \dim_x : \l_@@_the_array_box } } \end { minipage }
  \@@_create_extra_nodes:
  \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
  \bool_lazy_or:nnT { \int_compare_p:nNn \c@tabularnote > 0 } { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
  \@@_insert_tabularnotes:
}
\cs_new_protected:Npn \@@_insert_tabularnotes: {
  \skip_vertical:N 0.65ex
  \group_begin:
  \l_@@_notes_code_before_tl
  \tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }
  \int_compare:nNnT \c@tabularnote > 0
  { \begin { tabularnotes* } \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item \#1 } \strut
  \end { tabularnotes* } }
  \par}

The \TeX{} group is for potential specifications in the \l_@@_notes_code_before_tl.
\group_begin:
\l_@@_notes_code_before_tl
\tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }
\int_compare:nNnT \c@tabularnote > 0
{ \begin { tabularnotes* } \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item \#1 } \strut
  \end { tabularnotes* } }
\par

The following \par is mandatory for the event that the user has put \footnotesize (for example) in the notes/code-before.
The two dimensions `\aboverulesep` and `\heavyrulewidth` are parameters defined by `booktabs`.

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

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

Now, the general case.

We convert a value of `t` to a value of `1`.

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tma_int`.

```latex
\begin{tabular}{notes}
\unskip
\group_end:
\bool_if:NT \l_@@_notes_bottomrule_bool
\{
\bool_if:NTF \c_@@_booktabs_loaded_bool
\{
\The case of `baseline` equal to `b`. Remember that, when the key `b` is used, the `{array}` (of `array`) is constructed with the option `t` (and not `b`). Now, we do the translation to take into account the option `b`.

Now, the general case.

We convert a value of `t` to a value of `1`.

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tma_int`.

```
The command \@@_put_box_in_flow_bis: is used when the option delimiters/max-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.

We will compute the real width of both delimiters used.

Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.
The construction of the array in the environment \{NiceArrayWithDelims\} is, in fact, done by the environment \{@@-light-syntax\} or by the environment \{@@-normal-syntax\} (whether the option light-syntac is in force or not). When the key light-syntac is not used, the construction is a standard environment (and, thus, it’s possible to use verbatim in the array).

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

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

\NewDocumentEnvironment { @@-light-syntax } { b }
{ First, we test whether the environment is empty. It’s only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in \#1.

\tl_if_empty:nT { \#1 } { \@@_fatal:n { empty~environment } }
\tl_map_inline:nn { \#1 }
{ \str_if_eq:nnT { ##1 } { & } { \@@_fatal:n { ampersand~in~light-syntax } }
\str_if_eq:nnT { ##1 } { \\ } { \@@_fatal:n { double-backslash~in~light-syntax } }
}

Now, you extract the \CodeAfter 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.

\NewDocumentEnvironment { @@-light-syntax } { b }
{ First, we test whether the environment is empty. It’s only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in \#1.

\tl_if_empty:nT { \#1 } { \@@_fatal:n { empty~environment } }
\tl_map_inline:nn { \#1 }
{ \str_if_eq:nnT { ##1 } { & } { \@@_fatal:n { ampersand~in~light-syntax } }
\str_if_eq:nnT { ##1 } { \\ } { \@@_fatal:n { double-backslash~in~light-syntax } }
}

Now, you extract the \CodeAfter 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
{ }

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:Nnn \l_@@_end_of_row_tl { \l_@@_end_of_row_tl } \l_@@_end_of_row_tl
\exp_args:NN \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.

\begin{verbatim}
\int_compare:nNnT \l_@@_last_row_int = {-1}
\lbrace \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } \rbrace
\end{verbatim}

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

\begin{verbatim}
\exp_args:NV \@@_array: \g_@@_preamble_tl
\end{verbatim}

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

\begin{verbatim}
\seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
\exp_args:NV \@@_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
\end{verbatim}

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).

\begin{verbatim}
\cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
{ \str_if_eq:VnT \g_@@_name_env_str { #2 } { \@@_fatal:n { empty~environment } } \end{verbatim}

\begin{verbatim}
\end { #2 }
\end{verbatim}

The command \texttt{\textbackslash@\textbackslash create\_col\_nodes}: will construct a special last row. That last row is a false row used to create the \texttt{\textbackslash 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).

\begin{verbatim}
\cs_new:Npn \@@_create_col_nodes:
{ \crcr \int_compare:nNnT \l_@@_first_col_int = 0
{ \omit \hbox_overlap_left:n
\lbrace \bool_if:NT \l_@@_code_before_bool
{ \pgfsys@markposition { \@@_env: - col - 0 } } \pgfpicture
\pgfmemberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } } \endpgfpicture
\skip_horizontal:N 2\col@sep
\end{verbatim}

\texttt{\textbackslash crcr}
The following instruction must be put after the instruction \omit.
\bool_gset_true:N \g_@@_row_of_col_done_bool

First, we put a col node on the left of the first column (of course, we have to do that after the \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 }
\skip_horizontal:N 0.5\arrayrulewidth
}
}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\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 }
\skip_horizontal:N -0.5\arrayrulewidth
}
}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\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
}

We compute in \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 \halign and because we have to use this variable in other cells (of the same row). The affectation of \g_tmpa_skip, like all the affectations, must be done after the \omit of the cell.

We give a default value for \g_tmpa_skip (0 pt plus 1 fill) but it will just after be erased by a fixed value in the concerned cases.
\skip_gset:Nn \g_tmpa_skip { 0 pt plus 1 fill }
\bool_if:NT \l_@@_auto_columns_width_bool
{\dim_compare:nNnTF \l_@@_columns_width_dim > \c_zero_dim
{\bool_lazy_and:nnTF \l_@@_auto_columns_width_bool \l_@@_block_auto_columns_width_bool
{\skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim
\skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim
}
We begin a loop over the columns. The integer $\g_tmpa_int$ will be the number of the current column. This integer is used for the Tikz nodes.

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

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

\begin{verbatim}
\skip_horizontal:N \g_tmpa_skip
\bool_if:NT \l_@@_code_before_bool
 { \hbox
 { \skip_horizontal:N -0.5\arrayrulewidth
 \pgfsys@markposition { \@@_env: - col - 2 }
 \skip_horizontal:N 0.5\arrayrulewidth
 }
 }
\end{verbatim}

We create the col node on the right of the current column.

\begin{verbatim}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
\pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
\skip_horizontal:N \g_@@_width_last_col_dim
\end{verbatim}

\begin{verbatim}
\bool_if:NT \g_@@_last_col_found_bool
 { \hbox_overlap_right:n
 { \% \skip_horizontal:N \col@sep
 \skip_horizontal:N \g_@@_width_last_col_dim
 }\end{verbatim}
Here is the preamble for the “first column” (if the user uses the key first-col)
\tl_const:Nn \c_@@_preamble_first_col_tl
{ > }
\tl_const:Nn \c_@@_preamble_first_col_tl
{ > }

At the beginning of the cell, we link \CodeAfter to a command which do not begin with \omit (whereas the standard version of \CodeAfter begins with \omit).
\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
\bool_gset_true:N \g_@@_after_col_zero_bool
\@@_begin_of_row:

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

We insert \l_@@_code_for_first_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”:
\bool_lazy_and:nnT
{ \int_compare_p:nNn \c@iRow > 0 }
\bool_lazy_or_p:nn
{ \int_compare_p:nNn \l_@@_last_row_int < 0 }
\int_compare_p:nNn \c@iRow < \l_@@_last_row_int 

Be careful: despite this letter l the cells of the “first column” are composed in a R manner since they are composed in a \hbox_overlap_left:n.
We actualise the width of the “first column” because we will use this width after the construction of the array.

\dim_gset:Nn \g_@@_width_first_col_dim
\dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box }

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

\hbox_overlap_left:n
\dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
\@@_node_for_the_cell:
\box_use_drop:N \l_@@_cell_box
\skip_horizontal:N \l_@@_left_delim_dim
\skip_horizontal:N \l_@@_left_margin_dim
\skip_horizontal:N \l_@@_extra_left_margin_dim
\bool_gset_false:N \g_@@_empty_cell_bool
\skip_horizontal:N -2\col@sep

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

\tl_const:Nn \c_@@_preamble_last_col_tl
\c_@@_preamble_last_col_tl
\int_gincr:N \c@jCol
\int_gset_eq:NN \g_@@_col_total_int \c@jCol

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

\hbox_set:Nw \l_@@_cell_box
\@@_math_toggle_token:
\int_compare:nNnT \c@iRow > 0
\bool_lazy_or:nnTF \l_@@_last_row_int < 0
\int_compare_p:nNn \c@iRow < \l_@@_last_row_int
\l_@@_code_for_last_col_tl
\xglobal \colorlet { nicematrix-last-col } { . }

At the beginning of the cell, we link \CodeAfter to a command which do not begin with \omit (whereas the standard version of \CodeAfter begins with \omit).

\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n

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

\bool_gset_true:N \g_@@_last_col_found_bool
\int_gincr:N \c@jCol
\int_gset_eq:NN \g_@@_col_total_int \c@jCol

We insert \l_@@_code_for_last_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”.

\int_compare:nNnT \c@iRow > 0
\bool_lazy_or:nnT
\int_compare_p:nNn \l_@@_last_row_int < 0
\int_compare_p:nNn \c@iRow < \l_@@_last_row_int
\l_@@_code_for_last_col_tl
\xglobal \colorlet { nicematrix-last-col } { . }

We insert...
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

\begin{verbatim}
\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
\begin{NiceArray}
\use:c { #1 NiceArray }
\end{NiceArray}
\end{verbatim}

The following command will be linked to \NotEmpty in the environments of nicematrix.

\begin{verbatim}
\cs_new_protected:Npn \@@_NotEmpty:
{% \bool_gset_true:N \g_@@_not_empty_cell_bool }
\end{verbatim}

The environments \{NiceTabular\} and \{NiceTabular*\}

\begin{verbatim}
\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 }
\end{NiceArray}
\NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
\str_gset:Nn \g_@@_name_env_str { NiceTabular* }
\keys_set:nn { NiceMatrix / NiceTabular } { #1 , #4 }
% \bool_set_true:N \l_@@_NiceTabular_bool
\NiceArray { #3 }
\end{NiceArray}
\end{verbatim}

After the construction of the array

\begin{verbatim}
% \after_array:
% \group_begin:
\end{verbatim}

When the option 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 `\hbox_overlap_right:n`) but (if `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 `\Vdots` drawn in that last column. That’s why we fix the correct value of `\l_@@_last_col_int` in that case.

```latex
\bool_if:NT \g_@@_last_col_found_bool
{ \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }
```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we fix the real value of `\l_@@_last_col_int`.

```latex
\bool_if:NT \l_@@_last_col_without_value_bool
{ \dim_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int

\iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_col_ \int_use:N \g_@@_env_int }
{ \int_use:N \g_@@_col_total_int }
}
\str_if_empty:NF \l_@@_name_str
{ \iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_col_ \l_@@_name_str }
{ \int_use:N \g_@@_col_total_int }
}
}
\iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
```

It’s also time to give to `\l_@@_last_row_int` its real value. But, if the user had used the option `last-row` without value, we write in the `aux` file the number of that last row for the next run.

```latex
\bool_if:NT \l_@@_last_row_without_value_bool
{ \dim_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int

\iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
{ \int_use:N \g_@@_row_total_int }
}
\str_if_empty:NF \l_@@_name_str
{ \iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
{ \int_use:N \g_@@_row_total_int }
}
}
\iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
```

If the option `light-syntax` is used, we have nothing to write since, in this case, the number of rows is directly determined.

```latex
\bool_if:NT \l_@@_light_syntax_bool
{ \iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int }
{ \int_use:N \g_@@_row_total_int }
}
\str_if_empty:NF \l_@@_name_str
{ \iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { @@_last_row_ \l_@@_name_str }
{ \int_use:N \g_@@_row_total_int }
}
}
\iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
```

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

```latex
\bool_if:NT \l_@@_code_before_bool
{ \iow_now:Nn \@mainaux \ExplSyntaxOn
\iow_now:Nx \@mainaux
{ \seq_clear_new:c { @@_size_ \int_use:N \g_@@_env_int _ seq } }
```

105
If the user has used a key `last-row` in an environment with preamble (like `{NiceArray}`) and that last row has not been found, we have to increment the value because it will be decreased when used in the code-before.

```
\bool_lazy_and:nnTF
  { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
  { \bool_not_p:n \g_@@_last_col_found_bool }
@@_succ:n
\int_use:N \g_@@_col_total_int
```

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

```
\seq_gset_from_clist:cn { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int_seq }
\seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
```

```
\iow_now:Nn \@mainaux \ExplSyntaxOff
```

Now, you create the diagonal nodes by using the \row nodes and the \col nodes. If the engine is \xetex or \luatex we also create the “½ nodes”.

```
\@@_create_diag_nodes:
```

By default, the diagonal lines will be parallelized\textsuperscript{53}. There are two types of diagonals lines: the \Ddots diagonals and the \Iddots diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current \NiceArray environment.

```
\bool_if:NT \l_@@_parallelize_diags_bool
  {
    \int_gzero_new:N \g_@@_ddots_int
    \int_gzero_new:N \g_@@_iddots_int
    \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
  }
```

```
\int_zero_new:N \l_@@_initial_i_int
\int_zero_new:N \l_@@_initial_j_int
\int_zero_new:N \l_@@_final_i_int
\int_zero_new:N \l_@@_final_j_int
\bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
\if\pdffontdimen
```

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

```
\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 }
  }
```

\textsuperscript{53}It’s possible to use the option \texttt{parallelize-diags} to disable this parallelization.
The dimension \l_@@_xdots_shorten_dim corresponds to the option xdots/shorten available to the user. That's why we give a new value according to the current value, and not an absolute value.

\begin{verbatim}
\def\dim_set:Nn #1\l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
\end{verbatim}

Now, we actually draw the dotted lines (specified by \texttt{\textbackslash Cdots}, \texttt{\textbackslash Vdots}, etc.).

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

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it will do nothing. The corners are computed in \l_@@_empty_corner_cells_seq which will contain all the empty cells (and not in a block) considered in the corners of the array.

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

The sequence \g_@@_pos_of_blocks_seq must be “adjusted” (for the case where the user have written something like \texttt{\textbackslash Block{1-}}).

\begin{verbatim}
\@@_adjust_pos_of_blocks_seq:
\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 \texttt{\textbackslash Cdots}, \texttt{\textbackslash Vdots}, etc.) and no corners. In that case, we switch to a shortcut version of \texttt{\@@_vline_i:nn} and \texttt{\@@_hline:nn}.

\begin{verbatim}
\bool_lazy_all:nT
\end{verbatim}

When \texttt{light-syntax} is used, we insert systematically a \texttt{\textbackslash CodeAfter} in the flow. Thus, it’s possible to have two instructions \texttt{\textbackslash CodeAfter} and the second may be in \g_nicematrix_code_after_tl. That’s why we set \texttt{\textbackslash Code-after} to be no-op now.

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

We clear the list of the names of the potential \texttt{\textbackslash SubMatrix} that will appear in the \texttt{\textbackslash CodeAfter} (unfortunately, that list has to be global).

\begin{verbatim}
\seq_gclear:N \g_@@_submatrix_names_seq
\end{verbatim}
And here’s the \texttt{\textbackslash CodeAfter}. Since the \texttt{\textbackslash CodeAfter} may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command \texttt{\textbackslash@@_CodeAfter_keys}:

\begin{verbatim}
\exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
\scan_stop:
\tl_gclear:N \g_nicematrix_code_after_tl
\group_end:
\end{verbatim}

\texttt{\g_nicematrix_code_before_tl} is for instructions in the cells of the array such as \texttt{\rowcolor} and \texttt{\cellcolor} (when the key \texttt{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.

\begin{verbatim}
\tl_if_empty:NF \g_nicematrix_code_before_tl
\end{verbatim}

The command \texttt{\rowcolor} in tabular will in fact use \texttt{\rectanglecolor} in order to follow the behaviour of \texttt{\rowcolor} of \texttt{colortbl}. That’s why there may be a command \texttt{\rectanglecolor} in \texttt{\g_nicematrix_code_before_tl}. In order to avoid an error during the expansion, we define a protected version of \texttt{\rectanglecolor}.

\begin{verbatim}
\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 }
\exp_not:V \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:
\end{verbatim}

The command \texttt{\CT@arc@} contains the instruction of color for the rules of the array\textsuperscript{54}. This command is used by \texttt{\CT@arc@} but we use it also for compatibility with \texttt{colortbl}. But we want also to be able to use color for the rules of the array when \texttt{colortbl} is \textit{not} loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by \texttt{colortbl}.

\begin{verbatim}
\cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
\end{verbatim}

The following command will extract the potential options (between square brackets) at the beginning of the \texttt{\textbackslash CodeAfter} (that is to say, when \texttt{\textbackslash CodeAfter} is used, the options of that “command” \texttt{\textbackslash CodeAfter}).

\begin{verbatim}
\NewDocumentCommand \@@_CodeAfter_keys: { O { } }
\keys_set:nn { NiceMatrix / CodeAfter } { #1 }
\end{verbatim}

We remind that the first mandatory argument of the command \texttt{\Block} is the size of the block with the special format \texttt{i-j}. However, the user is allowed to omit \texttt{i} or \texttt{j} (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in \texttt{\g@@_pos_of_blocks_seq} (and \texttt{\g@@_blocks_seq}) as a number of rows (resp. columns) for the block equal to 100. It’s possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

\begin{verbatim}
\cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
\seq_gset_map:x:NNn \g@@_pos_of_blocks_seq \g@@_pos_of_blocks_seq
\exp_not:V \g_nicematrix_code_before_tl
\end{verbatim}

The following command must \textit{not} be protected.

\begin{verbatim}
\cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
\end{verbatim}

\textsuperscript{54}e.g. \texttt{\color\{rgb\}\{0.5,0.5,0\}}
We recall that, when externalization is used, \texttt{\textbackslash tikzpicture} and \texttt{\textbackslash endtikzpicture} (or \texttt{\textbackslash pgfpicture} and \texttt{\textbackslash endpgfpicture}) must be directly “visible”. That’s why we have to define the adequate version of \texttt{\textbackslash@@\_draw\_dotted\_lines}: whether Tikz is loaded or not (in that case, only \texttt{PGF} is loaded).

\begin{verbatim}
\AtBeginDocument
{\cs_new_protected:Npx \@@_draw_dotted_lines:
{\c_@@_pgfortikzpicture_tl
\@@_draw_dotted_lines_i:
\c_@@_endpgfortikzpicture_tl
}
}
\end{verbatim}

The following command \textit{must} be protected because it will appear in the construction of the command \texttt{\textbackslash@@\_draw\_dotted\_lines}:.

\begin{verbatim}
\cs_new_protected:Npn \@@_draw_dotted_lines_i:
{
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\g_@@_HVdotsfor_lines_tl
\g_@@_Vdots_lines_tl
\g_@@_Ddots_lines_tl
\g_@@_Iddots_lines_tl
\g_@@_Cdots_lines_tl
\g_@@_Ldots_lines_tl
}
\end{verbatim}

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

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

\begin{verbatim}
\cs_new_protected:Npn \@@_create_diag_nodes:
{\pgfpicture
\pgfrememberpicturepositiononpagetrue
\int_step_inline:nn { \int_max:nn \c@iRow \c@jCol + 1 }
{\@@_qpoint:n { row - \int_min:nn \c@iRow \l_@@_old_iRow_int }
\dim_set_eq:NN \l_tmpa_dim \pgf@y
\@@_qpoint:n { col - \int_min:nn \c@jCol \l_@@_old_jCol_int }
\pgfcoordinate { \@@_env: - ##1 } { \pgfpoint \pgf@x \l_tmpa_dim }
}\pgfnodealias
{\@@_env: - last }
}
\end{verbatim}
As for now, the $\frac{1}{2}$ nodes are not documented and they are created only in the native UTF8 engines, that is to say XeLaTeX and LuaLaTeX.

\begin{Verbatim}
\bool_if:nT { \sys_if_engine_xetex_p: || \sys_if_engine_luatex_p: }
\tl_put_right:Nn \@@_create_diag_nodes: \@@_create_half_nodes:
\cs_new_protected:Npn \@@_create_half_nodes:
\int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - ##1 ½ }
\pgfpointscale { 0.5 }
\pgfpointadd
{ \@@_qpoint:n { ##1 } }
{ \@@_qpoint:n { \int_eval:n { ##1 + 1 } } }
\endpgfpicture
\end{Verbatim}

We draw the dotted lines

A dotted line will be said \emph{open} in one of its extremities when it stops on the edge of the matrix and \emph{closed} otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

The command \texttt{\textbackslash \texttt{@@_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{\textbackslash \texttt{l_@@_initial_i_int}} and \texttt{\textbackslash \texttt{l_@@_initial_j_int}} which are the coordinates of one extremity of the line;
- \texttt{\textbackslash \texttt{l_@@_final_i_int}} and \texttt{\textbackslash \texttt{l_@@_final_j_int}} which are the coordinates of the other extremity of the line;
- \texttt{\textbackslash \texttt{l_@@_initial_open_bool}} and \texttt{\textbackslash \texttt{l_@@_final_open_bool}} to indicate whether the extremities are open or not.

\begin{Verbatim}
\cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
\{\null
\begin{Verbatim}
First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.
\end{Verbatim}
\end{Verbatim}

\begin{Verbatim}
\cs_set:cpn { \texttt{@@_dotted} } { #1 - #2 } \}
\end{Verbatim}
Initialization of variables.

\int_set:Nn \l_@@_initial_i_int { #1 }
\int_set:Nn \l_@@_initial_j_int { #2 }
\int_set:Nn \l_@@_final_i_int { #1 }
\int_set:Nn \l_@@_final_j_int { #2 }

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean \l_@@_stop_loop_bool will be used to control these loops. In the first loop, we search the “final” extremity of the line.

\bool_set_false:N \l_@@_stop_loop_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
{\int_add:Nn \l_@@_final_i_int { #3 }\int_add:Nn \l_@@_final_j_int { #4 }

We test if we are still in the matrix.

\bool_set_false:N \l_@@_final_open_bool
\int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
{ \int_compare:nNnTF \{ #3 \} = 1
\bool_set_true:N \l_@@_final_open_bool
}
\int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
{ \bool_set_true:N \l_@@_final_open_bool }
\int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
{ \int_compare:nNnT \{ #4 \} = { -1 }
\bool_set_true:N \l_@@_final_open_bool
}
\bool_if:NTF \l_@@_final_open_bool
If we are outside the matrix, we have found the extremity of the dotted line and it’s an open extremity.

We do a step backwards.

\int_sub:Nn \l_@@_final_i_int { #3 }
\int_sub:Nn \l_@@_final_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool

If we are in the matrix, we test whether the cell is empty. If it’s not the case, we stop the loop because we have found the correct values for \l_@@_final_i_int and \l_@@_final_j_int.

{ \cs_if_exist:cTF
  \l_@@_dotted_
  \int_use:N \l_@@_final_i_int -
  \int_use:N \l_@@_final_j_int
}
{ \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_final_open_bool
  \bool_set_true:N \l_@@_stop_loop_bool
}
If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

For \l_\@\_initial\_i\_int and \l_\@\_initial\_j\_int the programation is similar to the previous one.
\int_add:Nn \l_@@_initial_j_int { #4 } 
\bool_set_true:N \l_@@_stop_loop_bool 
}

\cs_if_exist:cTF
{
\begin{verbatim}
\int_use:N \l_@@_initial_i_int - 
\int_use:N \l_@@_initial_j_int 
\end{verbatim}
}
\int_add:Nn \l_@@_initial_i_int { #3 } 
\int_add:Nn \l_@@_initial_j_int { #4 } 
\bool_set_true:N \l_@@_initial_open_bool 
\bool_set_true:N \l_@@_stop_loop_bool 

\cs_if_exist:cTF
{
\begin{verbatim}
pgf@sh@ns@\@@_env:
- \int_use:N \l_@@_initial_i_int 
- \int_use:N \l_@@_initial_j_int 
\end{verbatim}
}
\bool_set_true:N \l_@@_stop_loop_bool 

\cs_set:cpn
{
\begin{verbatim}
\int_use:N \l_@@_initial_i_int - 
\int_use:N \l_@@_initial_j_int 
\end{verbatim}
}{ }

\\seq_gput_right:Nx \g_@@_pos_of_xdots_seq
{
\int_use:N \l_@@_initial_i_int 
\int_use:N \l_@@_initial_j_int 
\int_use:N \l_@@_final_i_int 
\int_use:N \l_@@_final_j_int 
}

\seq_map_inline:Nn \g_@@_submatrix_seq
 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }

\end{verbatim}

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.

\seq_gput_right:Nx \g_@@_pos_of_xdots_seq
{
\int_use:N \l_@@_initial_i_int 
\int_use:N \l_@@_initial_j_int 
\int_use:N \l_@@_final_i_int 
\int_use:N \l_@@_final_j_int 
}

The following command (when it will be written) will set the four counters \l_@@_row_min_int, \l_@@_row_max_int, \l_@@_col_min_int and \l_@@_col_max_int to the intersections of the submatrices which contains the cell of row \texttt{#1} and column \texttt{#2}. As of now, it’s only the whole array (excepted exterior row and columns).

\cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
{
\int_set:Nn \l_@@_row_min_int 1 
\int_set:Nn \l_@@_col_min_int 1 
\int_set_eq:NN \l_@@_row_max_int \c@iRow 
\int_set_eq:NN \l_@@_col_max_int \c@jCol 

\seq_map_inline:Nn \g_@@_submatrix_seq
 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }

We do a loop over all the submatrices specified in the code-before. We have stored the position of all those submatrices in \g_@@_submatrix_seq.
and \#2 are the numbers of row and columns of the cell where the command of dotted line (ex.: \Vdots) has been issued. \#3, \#4, \#5 and \#6 are the specification (in \(i\) and \(j\)) of the submatrix where are analysing.

\begin{verbatim}
\cs_set_protected:Npn \@\_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{\
  \bool_if:nTF\
    {\int_compare_p:n { #3 <= #1 } && \int_compare_p:n { #1 <= #5 } && \int_compare_p:n { #4 <= #2 } && \int_compare_p:n { #2 <= #6 } }\
  {\
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }\
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }\
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }\
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }\
  }\
}
\cs_new_protected:Npn \@\_set_initial_coords:\
{\
  \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x\
  \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y\
}
\cs_new_protected:Npn \@\_set_final_coords:\
{\
  \dim_set_eq:NN \l_@@_x_final_dim \pgf@x\
  \dim_set_eq:NN \l_@@_y_final_dim \pgf@y\
}
\cs_new_protected:Npn \@\_set_initial_coords_from_anchor:n #1\#1\
{\
  \pgfpointanchor\
  {\@@_env: - \int_use:N \l_@@_initial_i_int - \int_use:N \l_@@_initial_j_int}\
  \ @@_set_initial_coords:\
}
\cs_new_protected:Npn \@\_set_final_coords_from_anchor:n #1\#1\
{\
  \pgfpointanchor\
  {\@@_env: - \int_use:N \l_@@_final_i_int - \int_use:N \l_@@_final_j_int}\
  \ @@_set_final_coords:\
}
\cs_new_protected:Npn \@@_open_x_initial_dim:\n\{\
  \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim\
  \int_step_inline:nnnn \l_@@_first_row_int \g_@@_row_total_int \l_@@_col_int
  \{\
    \int_compare_p:n { \int_use:N \l_@@_col_int \@@_env: - \int_use:N \l_@@_col_int }\
    \pgfpointanchor\
    \@@_env: - \int_use:N \l_@@_initial_i_int - \int_use:N \l_@@_initial_j_int \{ west \}
  \}
\}
\cs_new_protected:Npn \@@_open_x_final_dim:\n\{\
  \dim_set_eq:NN \l_@@_x_final_dim \c_max_dim\
  \int_step_inline:nnnn \l_@@_first_row_int \g_@@_row_total_int \l_@@_col_int
  \{\
    \int_compare_p:n { \int_use:N \l_@@_col_int \@@_env: - \int_use:N \l_@@_col_int }\
    \pgfpointanchor\
    \@@_env: - \int_use:N \l_@@_final_i_int - \int_use:N \l_@@_final_j_int \{ west \}
  \}
\}
\end{verbatim}
If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

\dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
{ \@@_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 \col@sep
}

\cs_new_protected:Npn \@@_open_x_final_dim:
{ \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
  \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
  { \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
    { \pgfpointanchor { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int } { east }
      \dim_set:NN \l_@@_x_final_dim \pgf@x
    }
  }
}

\cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
{ \@@_adjust_to_submatrix:nn { #1 } { #2 }
  \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
  { \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1
    \group_begin:
    \int_compare:nNnTF { #1 } = 0
      { \color { nicematrix-first-row } }
      { \color { nicematrix-last-row } }
    \keys_set:nn { NiceMatrix / xdots } { #3 }
    \@@_actually_draw_Ldots:
    \group_end:
}

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.

\cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
{ \@@_adjust_to_submatrix:nn { #1 } { #2 }
  \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
  { \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1
    \group_begin:
    \int_compare:nNnTF { #1 } = 0
      { \color { nicematrix-first-row } }
      { \color { nicematrix-last-row } }
    \keys_set:nn { NiceMatrix / xdots } { #3 }
    \@@_actually_draw_Ldots:
    \group_end:
}

We remind that, when there is a “last row” \l_@@_last_row_int will always be (after the construction of the array) the number of that “last row” even if the option last-row has been used without value.
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.

\begin{verbatim}
\cs_new_protected:Npn \@@_actually_draw_Ldots: 
\bool_if:NTF \l_@@_initial_open_bool 
{ \@@_open_x_initial_dim: \@@_qpoint:n \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y 
\} \bool_if:NTF \l_@@_final_open_bool 
{ \@@_open_x_final_dim: \@@_qpoint:n \dim_set_eq:NN \l_@@_y_final_dim \pgf@y 
\}
\dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim 
\dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim 
\@@_draw_line:
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3 
\@@_adjust_to_submatrix:nn \cs_if_free:cT { @@ _ dotted _ #1 - #2 } 
\@@_find_extremities_of_line:nnnn \group_begin: 
\int_compare:nNnTF { #1 } = 0 
{ \color { nicematrix-first-row } } 
\int_compare:nNnT { #1 } = \l_@@_last_row_int 
{ \color { nicematrix-last-row } } 
\keys_set:nn \group_end:
\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{verbatim}
\@@_adjust_to_submatrix:nn \cs_if_free:cT \@@_find_extremities_of_line:nnnn \{ #1 \} \{ #2 \} 0 1
\group_begin:
\int_compare:nNnTF { #1 } = 0 
{ \color { nicematrix-first-row } } 
\int_compare:nNnT { #1 } = \l_@@_last_row_int 
{ \color { nicematrix-last-row } } 
\keys_set:nn \group_end:
\end{verbatim}

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_Cdots:nnn #1 #2 #3 
\@@_adjust_to_submatrix:nn \cs_if_free:cT \@@_find_extremities_of_line:nnnn \{ #1 \} \{ #2 \} 0 1
\group_begin:
\int_compare:nNnTF { #1 } = 0 
{ \color { nicematrix-first-row } } 
\int_compare:nNnT { #1 } = \l_@@_last_row_int 
{ \color { nicematrix-last-row } } 
\keys_set:nn \group_end:
\end{verbatim}

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 \texttt{\@@_actually_draw_Cdots:} 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}.
\cs_new_protected:Npn \@@_open_y_final_dim: \
\{ \\
\@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base } \\
\dim_set:Nn \l_@@_y_final_dim { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } \\
\int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int \
\{ \\
\cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 } \\
{ \\
\pgfpointanchor { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 } { south } \\
\dim_set:Nn \l_@@_y_final_dim { \dim_min:nn \l_@@_y_final_dim \pgf@y } 
\} 
\}
\}

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.

\cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3 \
\{ \\
\@@_adjust_to_submatrix:nn { #1 } { #2 } \\
\cs_if_free:cT { @@ _ dotted _ #1 - #2 } \\
{ \\
\@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 0 
\}

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\group_begin: \\
\int_compare:nNnTF { #2 } = 0 \\
{ \color { nicematrix-first-col } } \\
{ \\
\int_compare:nNnT { #2 } = \l_@@_last_col_int \\
{ \color { nicematrix-last-col } } 
\} 
\keys_set:nn { NiceMatrix / xdots } { #3 } \\
\tl_if_empty:VF \l_@@_xdots_color_tl \\
{ \color { \l_@@_xdots_color_tl } } \\
\@@_actually_draw_Vdots: \\
\group_end: 

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.

\cs_new_protected:Npn \@@_actually_draw_Vdots: \\
{ \\
The boolean \l_tmpa_bool indicates whether the column is of type l or may be considered as if.

\bool_set_false:N \l_tmmpa_bool
First the case when the line is closed on both ends.

```verbatim
\bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
{
  \@@_set_initial_coords_from_anchor:n { south\text{-}west }
  \@@_set_final_coords_from_anchor:n { north\text{-}west }
  \bool_set:Nn \l_tmpa_bool
  { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
}
```

Now, we try to determine whether the column is of type \textit{c} or may be considered as if.

```verbatim
\bool_if:NTF \l_@@_initial_open_bool
  \@@_open_y_initial_dim:
\bool_if:NTF \l_@@_final_open_bool
  \@@_open_y_final_dim:
\bool_if:NTF \l_@@_initial_open_bool
  {
    \bool_if:NTF \l_@@_final_open_bool
    {
      \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
      \dim_set_eq:NN \l_tmpa_dim \pgf@x
      \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
      \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    }
    \bool_if:NTF \l_@@_final_open_bool
    {
      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    }
  }

We may think that the final user won’t use a “last column” which contains only a command \texttt{\textbackslash vdots}. However, if the \texttt{\textbackslash 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.

```verbatim
\int_compare:nNnT \l_@@_last_col_int > \{-2\}
{
  \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
  {
    \dim_set_eq:NN \l_@@_right_margin_dim \l_@@_x_initial_dim
    \dim_set_eq:NN \l_@@_extra_right_margin_dim \l_@@_right_margin_dim
    \dim_set:Nn \l_@@_x_initial_dim \l_@@_x_final_dim
  }
}
```

Now the case where both extremities are closed. The first conditional tests whether the column is of type \textit{c} or may be considered as if.

```verbatim
\dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
{\dim_set:Nn \l_@@_x_initial_dim
  {\bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
    \l_@@_x_initial_dim \l_@@_x_final_dim}
  \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
}
```

```verbatim
\@@_draw_line:
```
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.

\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
{ \@@_adjust_to_submatrix:nn { #1 } { #2 }
\cs_if_free:cT { @@ _ dotted _ #1 - #2 }
{ \@@_find_extremities_of_line:mmnn { #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_Ddots:
\group_end:
}

The command \@@_actually_draw_Ddots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

\cs_new_protected:Npn \@@_actually_draw_Ddots:
{ \bool_if:NTF \l_@@_initial_open_bool
  { \@@_open_y_initial_dim:
    \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \@@_open_x_initial_dim:
  }
  \bool_if:NTF \l_@@_final_open_bool
  { \@@_open_y_final_dim:
    \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \@@_open_x_final_dim:
  }
  \@@_set_initial_coords_from_anchor:n { south-east } }

\bool_if:NTF \l_@@_final_open_bool
{ \bool_if:NTF \l_@@_final_coords_from_anchor:n { south-east } }

\bool_if:NTF \l_@@_parallelize_diags_bool
{ \int_gincr:N \g_@@_ddots_int
  \int_compare:nNnTF \g_@@_ddots_int = 1

We have retrieved the coordinates in the usual way (they are stored in \l_@@_x_initial_dim, etc.).

If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

\bool_if:NT \l_@@_parallelize_diags_bool
{ \int_gincr:N \g_@@_ddots_int
  \int_compare:nNnTF \g_@@_ddots_int = 1

We test if the diagonal line is the first one (the counter \g_@@_ddots_int is created for this usage).
If the diagonal line is the first one, we have no adjustment of the line to do but we store the $\Delta x$ and the $\Delta y$ of the line because these values will be used to draw the others diagonal lines parallels to the first one.

\begin{verbatim}
\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 $\ldots$ 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
\@@_adjust_to_submatrix:nn { #1 } { #2 }
\cs_if_free:cT { @@_dotted _#1 - #2 }
\@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
\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 $\@@_actually_draw_Iddots$: 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
\bool_if:NTF \l_@@_initial_open_bool
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
\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:
{\pgfrememberposition{\@@_draw_line:}}
We have to do a special construction with \texttt{\exp_args:NV} to be able to put in the list of options in the correct place in the Tikz instruction.

```latex
\cs_new_protected:Npn \@\@_draw_non_standard_dotted_line:nnnnnnnn #1 #2 #3 #4 #5
\begin {scope}
\exp_args:No \@\@_draw_non_standard_dotted_line:nV
\{ #1 \}
\exp_args:No \@\@_draw_non_standard_dotted_line:nV
\{ #2 \}
\exp_args:No \@\@_draw_non_standard_dotted_line:nV
\{ #3 \}
\exp_args:No \@\@_draw_non_standard_dotted_line:nV
\{ #4 \}
\exp_args:No \@\@_draw_non_standard_dotted_line:nV
\{ #5 \}
\end {scope}
```

We have used the fact that, in PGF, a color name can be put directly in a list of options (that’s why we have put directly \texttt{\l_@@_xdots_color_tl}).

The argument of \texttt{\@\@_draw_non_standard_dotted_line:n} is, in fact, the list of options.

```latex
\cs_new_protected:Npn \@\@_draw_non_standard_dotted_line:nnn #1 #2 #3
\begin {scope}
\draw
\[\]
shorten-> = \l_@@_xdots_shorten_dim ,
shorten<- = \l_@@_xdots_shorten_dim,
\]
( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
```

Be careful: We can’t put \texttt{\c_math_toggle_token} instead of $ in the following lines because we are in the contents of Tikz nodes (and they will be \texttt{rescanned} if the Tikz library babel is loaded).

```latex
\node [ sloped , above ] { $ \scriptstyle #2 $ }
\node [ sloped , below ] { $ \scriptstyle #3 $ }
\end {scope}
```

The command \texttt{\@\@_draw_standard_dotted_line:} draws the line with our system of dots (which gives a dotted line with real round dots).

```latex
\cs_new_protected:Npn \@\@_draw_standard_dotted_line:
\begin {scope}
\pgftransformshift
\pgfpointlineattime { 0.5 }
\pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim
\pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim
\pgftransformrotate
\begin {scope}
\fp_eval:n
\atand
\l_@@_y_final_dim - \l_@@_y_initial_dim ,
\l_@@_x_final_dim - \l_@@_x_initial_dim
\end {scope}
\end {scope}
```
The dimension \l_@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of expl3 to compute this length.

\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 } } }

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 aux file to say that one more compilation should be done.

\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:

The number of dots will be \l_tmpa_int + 1.
The dimensions \( \l_{\text{tmpa}} \) and \( \l_{\text{tmpb}} \) are the coordinates of the vector between two dots in the dotted line.

\[
\begin{align*}
\dim_set:Nn \l_{\text{tmpa}} \text{ and } \dim_set:Nn \l_{\text{tmpb}} \\
\{ \dim_gadd:Nn \l_{\text{tmpa}} \l_{\text{tmpb}}
\}
\end{align*}
\]

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 \( \l_{\text{tmpb}} \).

In the loop over the dots, the dimensions \( \l_{\text{x_initial}} \) and \( \l_{\text{y_initial}} \) 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 ofnicematrix 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 underscore activates _). That’s why these commands will be defined in a \AtBeginDocument and the arg spec will be rescanned.
End of the \AtBeginDocument.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

\keys_define:nn { NiceMatrix / Ddots }
{ draw-first .bool_set:N = \l_@@_draw_first_bool ,
draw-first .default:n = true ,
draw-first .value_forbidden:n = true }

The command \@@_Hspace: will be linked to \hspace in \{NiceArray\}.
\dos_new_protected:Npn \@@_Hspace:
{ \bool_gset_true:N \g_@@_empty_cell_bool
\hspace }

In the environment \{NiceArray\}, the command \multicolumn will be linked to the following command \@@_multicolumn:nnn.
\dos_set_eq:NN \@@_old_multicolumn \multicolumn
\dos_new:Npn \@@_multicolumn:nnn #1 #2 #3
{ We have to act in an expandable way since it will begin by a \multicolumn.
\exp_args:NNe \@@_old_multicolumn
{ #1 }

We will have to replace \tl_lower_case:n in the future since it seems to be deprecated.
\dos_new:Nne \str_case:nn { \str_foldcase:n { #2 } }
{ 1 { > \@@_Cell: l < \@@_end_Cell: } 
 r { > \@@_Cell: r < \@@_end_Cell: } 
 c { > \@@_Cell: c < \@@_end_Cell: } 
 { l | } { > \@@_Cell: l < \@@_end_Cell: | } 
 { r | } { > \@@_Cell: r < \@@_end_Cell: | } 
 { c | } { > \@@_Cell: c < \@@_end_Cell: | } 
 { | l } { | > \@@_Cell: l < \@@_end_Cell: } 
 { | r } { | > \@@_Cell: r < \@@_end_Cell: } 
 { | c } { | > \@@_Cell: c < \@@_end_Cell: } 
 { | l | } { | > \@@_Cell: l < \@@_end_Cell: | } 
 { | r | } { | > \@@_Cell: r < \@@_end_Cell: | } 
 { | c | } { | > \@@_Cell: c < \@@_end_Cell: | } 
}
The `\peek_remove_spaces:n` is mandatory.

\peek_remove_spaces:n
{
  \int_compare:nNnT #1 > 1
  {
    \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
      \int_use:N \c@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_use:N \c@iRow
      \int_eval:n { \c@jCol + #1 - 1 }
    }
    \int_gadd:Nn \c@jCol { #1 - 1 }
    \int_compare:nNnT \c@jCol > \g_@@_col_total_int
    { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
  }

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:
{
  \bool_lazy_and:nnTF
    { \int_compare_p:nNn \c@jCol = 0 }
    { \int_compare_p:nNn \l_@@_first_col_int = 0 }
  {
    \bool_if:NTF \g_@@_after_col_zero_bool
      {
        \multicolumn { 1 } { c } { }
        \@@_Hdotsfor_i
      }
      { \@@_fatal:n { Hdotsfor~in~col~0 } }
    }
  {
    \multicolumn { 1 } { c } { }
    \@@_Hdotsfor_i
  }
}

The command `\@@_Hdotsfor_i` is defined with `\NewDocumentCommand` 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
  We don’t put ! before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.
  \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
  {
    \tl_gput_right:Nx \g_@@_Hdotsfor_lines_tl
    \@@_Hdotsfor:nnnn
}

129
For the row, it’s easy.
\int_set:Nn \l_@@_initial_i_int { #1 }
\int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

For the column, it’s a bit more complicated.
\int_compare:nNnTF #2 = 1
{ \int_set:Nn \l_@@_initial_j_int 1
  \bool_set_true:N \l_@@_initial_open_bool
} { \cs_if_exist:cTF
  \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int
  - \int_eval:n { #2 - 1 }
} { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }

\int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
{ \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
  \bool_set_true:N \l_@@_final_open_bool
} { \cs_if_exist:cTF
  \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int
  - \int_eval:n { #2 + #3 }
} { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }

\group_begin:
\int_compare:nNnTF { #1 } = 0
{ \color { nicematrix-first-row } }

\end{AtBeginDocument}
We declare all the cells concerned by the \texttt{\textbackslash hdotsfor} as “dotted” (for the dotted lines created by \texttt{\textbackslash Dots}, \texttt{\textbackslash Dots}, etc., this job is done by \texttt{\textbackslash@@\_find\_extremities\_of\_line:nnnn}). This declaration is done by defining a special control sequence (to nil).

\texttt{\begin{verbatim}
\int_step_inline:nnn { #2 } { #2 + #3 - 1 }\
\cs_set:cpn { \@@\_dotted\_ #1 - ##1 } { }\
\end{verbatim}}

\texttt{\AtBeginDocument}\texttt{\begin{verbatim}
\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 } { #3 , #4 , down = \exp_not:n { #4 } , up = \exp_not:n { #5 } } } }\Endof\AtBeginDocument.\end{verbatim}}

\texttt{\begin{verbatim}
\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 { \tl_set:Nn \l_@@_initial_j_int { #2 } { \tl_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int \cs_if_exist:cTF \pgf @ sh @ ns @ \textbackslash@@\textbackslash env: \int_eval:n { #1 - 1 } \int_use:N \l_@@_initial_j_int \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl { \@@_Vdotsfor:nnnn { \int_use:N \c@iRow } { \int_use:N \c@jCol } { \#2 } { \#3 , \#4 , down = \exp_not:n { \#4 } , up = \exp_not:n { \#5 } } } }\end{verbatim}}

\texttt{\begin{verbatim}
\int_set:Nn \l_@@_initial_i_int { #1 - 1 } \bool_set_true:N \l_@@_initial_open_bool \cs_if_exist:cTF \pgf @ sh @ ns @ \textbackslash@@\textbackslash env: \int_eval:n { #1 - 1 } \int_use:N \l_@@_initial_i_int \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl { \@@_Vdotsfor:nnnn { \int_use:N \c@iRow } { \int_use:N \c@jCol } { \#1 , \#3 , \#4 , down = \exp_not:n { \#4 } , up = \exp_not:n { \#5 } } }\end{verbatim}}
We declare all the cells concerned by the \( \text{Vdots} \) as “dotted” (for the dotted lines created by \( \text{Cdots} \), \( \text{Ldots} \), etc., this job is done by \( \@@\_\text{find_extremities_of_line} \)). This declaration is done by defining a special control sequence (to nil).

\begin{verbatim}
\int_step_inline:nnn { #1 } { #1 + #3 - 1 } { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
\end{verbatim}

The command \( \@@\_\text{rotate} \) will be linked to \texttt{rotate} in \{\texttt{NiceArrayWithDelims}\}.\footnote{Indeed, we want that the user may use the command \texttt{line} in \texttt{CodeAfter} with \LaTeX\ counters in the arguments — with the command \texttt{value}.}

\begin{verbatim}
\cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }
\end{verbatim}

The command \texttt{\line} accessible in code-after

In the \texttt{CodeAfter}, the command \texttt{\@@\_line:nn} will be linked to \texttt{\line}. This command takes two arguments which are the specifications of two cells in the array (in the format \texttt{i-j}) and draws a dotted line between these cells. First, we write a command with an argument of the format \texttt{i-j} and applies the command \texttt{\int\_eval:n} to \texttt{i} and \texttt{j} ; this must not be protected (and is, of course fully expandable).\footnote{Indeed, we want that the user may use the command \texttt{line} in \texttt{CodeAfter} with \LaTeX\ counters in the arguments — with the command \texttt{value}.}

\begin{verbatim}
\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
{ \int_eval:n { #1 } - \int_eval:n { #2 } }
\end{verbatim}

With the following construction, the command \texttt{\@@\_double\_int\_eval:n} is applied to both arguments before the application of \texttt{\@@\_line\_i:nn} (the construction uses the fact the \texttt{\@@\_line\_i:nn} is protected and that \texttt{\@@\_double\_int\_eval:n} is fully expandable).
The following command must be protected (it’s used in the construction of \@@_draw_line_iii:nn).

\AtBeginDocument
{\cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
{
\pgfrememberpicturepositiononpagetrue
\pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
\dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
\pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
\dim_set_eq:NN \l_@@_x_final_dim \pgf@x
\dim_set_eq:NN \l_@@_y_final_dim \pgf@y
\@@_draw_line:
}}

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:.

\AtBeginDocument
{\cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
{\@@_draw_line:
}}

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

We want to avoid the thin white lines that are shown in some PDF viewers (e.g., with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction \texttt{\pgfusepath{fill}} (and they will be in the same instruction \texttt{\color{-}} in the resulting PDF).

The commands \texttt{\@@_rowcolor}, \texttt{\@@_columncolor} and \texttt{\@@_rectanglecolor} (which are linked to \texttt{\rowcolor}, \texttt{\columncolor} and \texttt{\rectanglecolor} before the execution of the \texttt{\CodeBefore}) don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence \texttt{\g_@@_colors_seq} will be built containing all the colors used by at least one of these instructions. Each color may be prefixed by its color model (e.g., \texttt{\[gray\]}{0.5}).

- For the color whose index in \texttt{\g_@@_colors_seq} is equal to 1, a list of instructions which use that color will be constructed in the token list \texttt{\g_@@_color_1_tl}. In that token list, the instructions will be written using \texttt{\@@_rowcolor:n}, \texttt{\@@_columncolor:n} and \texttt{\@@_rectanglecolor:nn} (corresponding to \texttt{\rowcolor}, \texttt{\columncolor} and \texttt{\rectanglecolor}).

bigskip

First, we look for the number of the color and, if it’s found, we store it in \texttt{\l_tmpa_int}. If the color is not present in \texttt{\l_@@_colors_seq}, \texttt{\l_tmpa_int} will remain equal to 0.

\begin{verbatim}
\int_zero:N \l_tmpa_int
\seq_map_indexed_inline:Nn \g_@@_colors_seq { \tl_if_eq:nnT { \l_tmpa_int } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } } \int_compare:nNnTF \l_tmpa_int = \c_zero_int
\end{verbatim}

First, the case where the color is a new color (not in the sequence).

\begin{verbatim}
\seq_gput_right:Nn \g_@@_colors_seq { \l_tmpa_int }\tl_gset:cx { g_@@_color_{ \seq_count:N \g_@@_colors_seq }_tl } { \l_tmpa_int } \tl_gclear:c { g_@@_color_{ \seq_count:N \g_@@_colors_seq }_tl }
\end{verbatim}

Now, the case where the color is not a new color (the color is in the sequence at the position \texttt{\l_tmpa_int}).

\begin{verbatim}
\tl_gput_right:cx { g_@@_color_ \int_use:N \l_tmpa_int_t1 } { \l_tmpa_int_t1 } \tl_gclear:c { g_@@_color_ \int_use:N \l_tmpa_int_t1 }
\end{verbatim}

The macro \texttt{\@@_actually_color}: will actually fill all the rectangles, color by color (using the sequence \texttt{\l_@@_colors_seq} and all the token lists of the form \texttt{\l_@@_color_i_t1}).

\begin{verbatim}
\cs_new_protected:Npn \@@_actually_color:
\pgfpicture \pgf@relevantforpicturesizefalse \seq_map_indexed_inline:Nn \g_@@_colors_seq { \color{\l_tmpa_int_t1} \use:c { g_@@_color_ \int_use:N \l_tmpa_int_t1 } \tl_gclear:c { g_@@_color_ \int_use:N \l_tmpa_int_t1 } \pgfusepath{fill} \endpgfpicture
\end{verbatim}
\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 }
}

Here is an example: \@@_rowcolor {red!15} {1,3,5-7,10-}
\NewDocumentCommand \@@_rowcolor { O { } m m }
\tl_if_blank:nF { #2 }
\@@_add_to_colors_seq:xn
\tl_if_blank:nF { #1 } { [ #1 ] } { #2 }
\@@_rowcolor:n { #3 }
\cs_new_protected:Npn \@@_rowcolor:n #1
\tl_set:Nn \l_@@_rows_tl { #1 }
\tl_set:Nn \l_@@_cols_tl { - }
\@@_cartesian_path:

Here is an example: \@@_columncolor:nn {red!15} {1,3,5-7,10-}
\NewDocumentCommand \@@_columncolor { O { } m m }
\tl_if_blank:nF { #2 }
\@@_add_to_colors_seq:xn
\tl_if_blank:nF { #1 } { [ #1 ] } { #2 }
\@@_columncolor:n { #3 }
\cs_new_protected:Npn \@@_columncolor:n #1
\tl_set:Nn \l_@@_rows_tl { - }
\tl_set:Nn \l_@@_cols_tl { #1 }
\@@_cartesian_path:

Here is an example: \@@_rectanglecolor{red!15}{2-3}{5-6}
\NewDocumentCommand \@@_rectanglecolor { O { } m m m }
\tl_if_blank:nF { #2 }
\@@_add_to_colors_seq:xn
\tl_if_blank:nF { #1 } { [ #1 ] } { #2 }
\@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt }
The last argument is the radius of the corners of the rectangle.

```
\NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
\begin{Verbatim}
{ \tl_if_blank:nF { #2 }
{ \@@_add_to_colors_seq:xn
  \begin{Verbatim}
  { \tl_if_blank:nF { #1 } \{ #1 \} \{ #2 \}
  \end{Verbatim}
  { \@@_rectanglecolor:nnn \{ #3 \} \{ #4 \} \{ #5 \} }
} }
\end{Verbatim}
\end{Verbatim}
```

The last argument is the radius of the corners of the rectangle.

```
\NewDocumentCommand \@@_roundedrectangle:n { O { } m m m m }
\begin{Verbatim}
{ \tl_if_blank:nF { #2 }
\begin{Verbatim}
  \@@_add_to_colors_seq:xn
\begin{Verbatim}
  \{ \tl_if_blank:nF { #1 } \{ #1 \} \{ #2 \}
\end{Verbatim}
  \{ \@@_roundedrectangle:nnn \{ #3 \} \{ #4 \} \{ #5 \} }
\end{Verbatim}
} }
\end{Verbatim}
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
\@@_cartesian_path:n \{ #3 \}
```

Here is an example:
```
\@@_cellcolor[rgb]\{0.5,0.5,0\}\{2-3,4-5,6-7,8-9,10-11,12-13,14-15,16-17\}
```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is an optional argument which a list of pairs key-value. As for now, there is only one key: `except-corners`. When that key is used, the cells in the corners are not colored.

```
\NewDocumentCommand \@@_arraycolor \{ O { } m O { } \}
\begin{Verbatim}
\keys_set:nn { NiceMatrix / arraycolor } \{ #3 \}
\end{Verbatim}
```

136
If the key `except-corners` is not used, it’s easy: we only have a rectangle to fill.

The interesting case is when the key `except-corners` is used. In that case, we can’t fill now the tabular because we don’t know the list of the cells of the corners. That’s why we postpone the treatment in the `\CodeAfter`.

The following command will be used in the `\CodeAfter`. #1 is the color model, #2 the color and #3 the value of the key `except-corners` (a sublist of `{NW,NE,SW,SE}).

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].`
#1 (optional) is the color space ; #2 is a list of intervals of rows ; #3 is the first color ; #4 is the second color ; #5 is for the optional list of pairs key-value.

\NewDocumentCommand \@@_rowcolors { O { } m m m O { } } { \group_begin: \tl_clear_new:N \l_@@_cols_tl \tl_set:Nn \l_@@_cols_tl { - } \keys_set:nn { NiceMatrix / rowcolors } { #5 } The boolean \l_@@_rowcolors_restart_bool will indicate whereas we are in a row of the first color or of the second color.

\bool_set_true:N \l_@@_rowcolors_restart_bool \[ #1 \]
\bool_lazy_and:nnT \l_@@_rowcolors_restart_bool { \cs_if_exist_p:c { c_@@_pos_of_blocks_ \int_use:N \g_@@_env_int _ seq } } \\int_set_eq:NN \l_tmpb_int \l_tmpa_int \If the key respect-blocks is in force, we have to adjust that value (of course).
Now, the last row of the block is computed in $l_{\text{tmpb}}$.

\begin{verbatim}
\tl_set:Nx \l_@@_rows_tl
{ \int_use:N \l_tmwa_int - \int_use:N \l_tmwb_int }
\bool_if:NTF \l_tmwa_bool
{ \tl_if_blank:nF { #3 } }
{ \tl_if_empty:nTF { #1 } \color{ #1 } }
{ #3 }
\endpgfpicture
\group_end:
\cs_new_protected:Npn \@@_rowcolors_i:nnnn #1 #2 #3 #4
{ \int_compare:nNnT { #3 } > \l_tmpb_int
{ \int_set:Nn \l_tmpb_int { \l_tmwb_int + 1 } }
\endpgfpicture\group_end:
\endverbatim

The following command return true when the block intersects the row $l_{\text{tmpa}}$.

\begin{verbatim}
\bool_if:NTF \@@_intersect_our_row:nnnn #1 #2 #3 #4
{ \int_compare:nNnT { #3 } > \l_tmwa_int
{ \int_set:Nn \l_tmwa_int { #3 } }
\prg_new_conditional:Nnn \@@_not_in_exterior:nnnn p
{ \bool_lazy_or:nnTF
{ \int_compare_p:nNn { #4 } = \c_zero_int }
{ \int_compare_p:nNn { #2 } = { \@@_succ:n { \c@jCol } } }
\prg_return_false:
\prg_return_true:
\endverbatim

The command $\@@_{\text{cartesian}}_{\text{path}}$: takes in two implicit arguments: $l_{\@\@_cols}$ and $l_{\@\@_rows}$.

\begin{verbatim}
\@@_{\text{cartesian}}_{\text{path}}:
\pgfusepath { fill }
\bool_set_true:N \l_tmwa_bool
\endverbatim
The following command uses two implicit arguments: \l_@@_rows_tl and \l_@@_cols_tl which are specifications for a set of rows and a set of columns. It creates a path but does not fill it. It must be filled by another command after. The argument is the radius of the corners.

We begin the loop over the columns.

We begin the loop over the rows.

Now, the numbers of both rows are in \l_tmpa_tl and \l_tmpb_tl.
When the user uses the key \texttt{colortbl}-like, the following command will be linked to \texttt{\cellcolor} in the tabular.

\NewDocumentCommand \@@_cellcolor_tabular { O { } m } {
  \tl_gput_right:Nx \g_nicematrix_code_before_tl {
    \cellcolor [ #1 ] { \exp_not:n { #2 } } { \int_use:N \c@iRow - \int_use:N \c@jCol } }
}

We must not expand the color (#2) because the color may contain the token \texttt{!} which may be activated by some packages (ex.: \texttt{babel} with the option \texttt{french} on latex and pdflatex).

\NewDocumentCommand \@@_rowcolor_tabular { O { } m } {
  \tl_gput_right:Nx \g_nicematrix_code_before_tl {
    \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } } {
      { \int_use:N \c@iRow - \int_use:N \c@jCol } {
        \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
    }
  }
}

\NewDocumentCommand \@@_columncolor_preamble { O { } m } {
  \int_compare:nNnT \c@iRow = 1 {
    You use \texttt{gput_left} because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells).
  }
  \tl_gput_left:Nx \g_nicematrix_code_before_tl {
    \exp_not:N \columncolor [ #1 ] { \exp_not:n { #2 } } {
      { \int_use:N \c@iRow - \int_use:N \c@jCol } }
  }
}

The vertical rules

We give to the user the possibility to define new types of columns (with \texttt{\newcolumntype} of \texttt{array}) for special vertical rules (\textit{e.g.} rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

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

\cs_set_eq:NN \OnlyMainNiceMatrix \use:n

141
Another definition of \OnlyMainNiceMatrix will be linked to the command in the environments of nicematrix. Here is that definition, called \@@_OnlyMainNiceMatrix:n.

\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
\{
\int_compare:nNnTF \l_@@_first_col_int = 0
\{ \@@_OnlyMainNiceMatrix_i:n { #1 } \}
\}
\int_compare:nNnF \c@jCol = 0
\{
\int_compare:nNnF \c@iRow = \l_@@_last_row_int - 1 \{ #1 \}
\}
\int_compare:nNnF \c@iRow = \l_@@_last_row_int \{ #1 \}
\}
\}

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.

\cs_new_protected:Npn \@@_vline:nn #1 #2
\{
\int_compare:nNnT { #1 } < { \c@jCol + 2 }
\{ \pgfpicture \@@_vline_i:nn { #1 } { #2 } \endpgfpicture \}
\}
\cs_new_protected:Npn \@@_vline_i:nn #1 #2
\{
\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_tmpc_tl.
\tl_set:Nx \l_tmpb_tl { #1 }
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@iRow \l_tmpa_tl
\{
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq \@@_test_vline_in_block:nnnn ##1
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq \@@_test_vline_in_block:nnnn ##1
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
\}

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_tmpc_tl.
\tl_set:Nx \l_tmpb_tl \{ #1 \}
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@iRow \l_tmpa_tl
\{
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq \@@_test_vline_in_block:nnnn ##1
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq \@@_test_vline_in_block:nnnn ##1
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
\}

The boolean \g_tmpa_bool indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by \Block or a virtual block corresponding to a dotted line, created by \Cdots, \Vdots, etc.), we will set \g_tmpa_bool to false and the small vertical rule won’t be drawn.
4018 \{ \@_test_vline_in_stroken_block:nnnn #1 \}
4019 \clist_if_empty:NT \l_@@_except_corners_clist
4020 \@_test_in_corner_v:
4021 \bool_if:NTF \g_tmpa_bool
4022 \{ \tl_if_empty:NT \l_tmpc_tl
4023 We keep in memory that we have a rule to draw.
4024 \{ \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl \}
4025 \{ \tl_if_empty:NT \l_tmpc_tl \}
4026 \@_vline_ii:nnnn \{ #1 \} \{ #2 \} \l_tmpc_tl \{ \int_eval:n \{\l_tmpa_tl - 1 \} \}
4027 \tl_clear:N \l_tmpc_tl \}
4028 \{ \tl_if_empty:NF \l_tmpc_tl \}
4029 \@_vline_ii:nnnn \{ #1 \} \{ #2 \} \l_tmpc_tl \{ \int_use:N \c@iRow \}
4030 \tl_clear:N \l_tmpc_tl \}
4031 \} \}
4032 \tl_if_empty:NF \l_tmpc_tl \}
4033 \@_vline_ii:nnnn \{ #1 \} \{ #2 \} \l_tmpc_tl \{ \int_eval:n \{\l_tmpa_tl - \@@_pred:n \l_tmpb_tl \} \}
4034 \tl_clear:N \l_tmpc_tl \}
4035 \} \}
4036 \cs_new_protected:Npn \@@_test_in_corner_v:
4037 \{ \int_compare:nNnTF \l_tmpb_tl = \{ \@@_succ:n \c@jCol \} \}
4038 \{ \l_@@_empty_corner_cells_seq \}
4039 \{ \l_tmpa_tl - \@@_pred:n \l_tmpb_tl \}
4040 \{ \bool_set_false:N \g_tmpa_bool \}
4041 \}
4042 \}
4043 \seq_if_in:NxT \l_@@_empty_corner_cells_seq
4044 \{ \l_tmpa_tl - \l_tmpb_tl \}
4045 \{ \int_compare:nNnTF \l_tmpb_tl = 1 \}
4046 \{ \bool_set_false:N \g_tmpa_bool \}
4047 \}
4048 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4

#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 numbers of the rows between which the rule has to be drawn.
The following command draws a complete vertical rule in the column \#1 (\#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key except-corners is not used).

\cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
{ \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

The command \@@_draw_hlines: draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as \Cdots) and in the corners (if the key except-corners is used).

\cs_new_protected:Npn \@@_draw_vlines:
{ \int_step_inline:nnn { \bool_if:NTF \l_@@_NiceArray_bool 1 2 } { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol } { \tl_if_eq:NnF \l_@@_vlines_clist { all } { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } } { \@@_vline:nn { ##1 } 1 } \\l_@@_vline:nnn { #1 } { #2 } 1 { \int_use:N \c@iRow } } }
The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn before the row #1. #2 is the number of consecutive occurrences of \Hline.

```latex
\cs_new_protected:Npn \@@_hline:nn #1 #2
\begin{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. When we have found a column corresponding to a rule to draw, we note its number in \l_tmpc_tl.
\tl_set:Nn \l_tmpa_tl { #1 }
\tl_clear_new:N \l_tmpc_tl
\int_step_variable:nNn \c@jCol \l_tmpb_tl
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq
{ \@@_test_hline_in_block:nnnn ##1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq
{ \@@_test_hline_in_block:nnnn ##1 }
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
{ \@@_test_hline_in_stroken_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
We keep in memory that we have a rule to draw.
\tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
\tl_if_empty:NT \l_tmpc_tl
\tl_clear:N \l_tmpc_tl
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq
{ \@@_test_hline_in_block:nnnn #1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq
{ \@@_test_hline_in_block:nnnn #1 }
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
{ \@@_test_hline_in_stroken_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 - 1 }
\tl_clear:N \l_tmpc_tl
\tl_if_empty:NT \l_tmpc_tl
\tl_set_eq:NN \l_tmpb_tl \l_tmpc_tl
```
```
\cs_new_protected:Npn \00_test_in_corner_h:
  {\int_compare:nNnTF \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 }
  \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 \l_tmpa_tl \l_tmpb_tl }
  \int_compare:nNnTF \l_tmpa_tl = \l_tmpb_tl - \l_tmpc_tl }
    \bool_set_false:N \g_tmpa_bool }
\seq_if_in:NxT \l_@@_empty_corner_cells_seq \l_tmpa_tl \l_tmpb_tl }
    \bool_set_false:N \g_tmpa_bool }
\bool_lazy_and:nnT \int_compare_p:nNn { #2 } > 1 \tl_if_blank_p:V \CT@drsc@ 
  \group_begin:
  \CT@drsc@
  \dim_set_eq:NN \l_tmpd_dim \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 )
  \pgfpathrectanglecorners \pgfpoint \l_tmpa_dim \l_tmpb_dim
  \pgfusepathqfill
  \group_end:
  \pgfsetlinewidth { 1.1 \arrayrulewidth }
#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.
\cs_new_protected:Npn \00_hline_ii:nnnn #1 #2 #3 #4
  {\pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - #3 }
    \dim_set_eq:NN \l_tmpa_dim \pgf@x
    \@@_qpoint:n { row - #1 }
    \dim_set_eq:NN \l_tmpb_dim \pgf@y
    \@@_qpoint:n { col - \l_@@_succ:n { #4 } }
    \dim_set_eq:NN \l_tmpc_dim \l_tmpb_dim
    \bool_lazy_and:nnT \int_compare_p:nNn { #2 } > 1 \tl_if_blank_p:V \CT@drsc@ 
    \group_begin:
    \CT@drsc@
    \dim_set:Nn \l_tmpd_dim \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 )
    \pgfpathrectanglecorners \pgfpoint \l_tmpa_dim \l_tmpb_dim
    \pgfpoint \l_tmpc_dim \l_tmpb_dim
    \pgfusepathqfill
    \group_end:
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \CT@drsc@
    \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
    \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
    \prg_replicate:nn { #2 - 1 }
    \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
    \dim_sub:Nn \l_tmpb_dim \doublerulesep
    \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
    \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
    \CT@arc@
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
\pgfsetrectcap
\pgfusepathqstroke
}

\cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
{ \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

The command \@@_draw_hlines: draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as \Cdots and in the corners (if the key except-corners is used).
\cs_new_protected:Npn \@@_draw_hlines:
{
\int_step_inline:nnn
{ \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
{ \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
{ \tl_if_eq:NnF \l_@@_hlines_clist { all }
{ \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
{ \@@_hline:nn { ##1 } 1 }
}
}

The command \@@_Hline: will be linked to \Hline in the environments of nicematrix.
\cs_set:Npn \@@_Hline: { \noalign { \ifnum 0 = `} \fi \@@_Hline_i:n { 1 } }

The argument of the command \@@_Hline_i:n is the number of successive \Hline found.
\cs_set:Npn \@@_Hline_i:n #1
{
\peek_meaning_ignore_spaces:NTF \Hline
{ \@@_Hline_ii:nn { #1 + 1 } }
{ \@@_Hline_iii:n { #1 } }
}
\cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
\cs_set:Npn \@@_Hline_iii:n #1
{
\skip_vertical:n
{\arrayrulewidth * ( #1 )
+ \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
}\tl_gput_right:Nx \g_@@_internal_code_after_tl
{ \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
\ifnum 0 = `{ \fi }
}

The key hvlines
The following command tests whether the current position in the array (given by \l_tmpa_tl for the row and \l_tmpb_tl for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean \l_tmpa_bool is set to false.
\cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
{\bool_lazy_all:nT
{ \int_compare_p:nNn \l_tmpa_tl > { #1 } }
{ \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
{ \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
{ \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
}
The same for vertical rules.

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

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

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

The key except-corners

When the key except-corners is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

\cs_new_protected:Npn \@@_compute_corners:
{\seq_clear_new:N \l_@@_empty_corner_cells_seq
\clist_map_inline:Nn \l_@@_except_corners_clist
{\str_case:nnF {##1}
{NW}{\@@_compute_a_corner:nnnnnnnnnn 1 1 1 \c@iRow \c@jCol}
{NE}{\@@_compute_a_corner:nnnnnnnn 1 \c@jCol 1 \c@iRow 1}
{SW}{\@@_compute_a_corner:nnnnnnnnnn 1 \c@iRow 1 \c@jCol 1}
{SE}{\@@_compute_a_corner:nnnnnnnnnn 1 1 1 \c@iRow \c@jCol}
}
}

The sequence \l_@@_empty_corner_cells_seq will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

\seq_clear_new:N \l_@@_empty_corner_cells_seq
\clist_map_inline:Nn \l_@@_except_corners_clist
{\str_case:nnF {##1}
{NW}{\@@_compute_a_corner:nnnnnnnnnn 1 1 1 \c@iRow \c@jCol}
{NE}{\@@_compute_a_corner:nnnnnnnnnn 1 \c@jCol 1 \c@iRow 1}
{SW}{\@@_compute_a_corner:nnnnnnnnnn 1 \c@iRow 1 \c@jCol 1}
{SE}{\@@_compute_a_corner:nnnnnnnnnn 1 1 1 \c@iRow \c@jCol}
}
“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 \l_@@_empty_corner_cells_seq.

The six arguments of \@@_compute_a_corner:nnnnnn are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

\cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6

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_tmpa_bool will be raised when a non-empty cell is found.

\bool_set_false:N \l_tmpa_bool \int_zero_new:N \l_@@_last_empty_row_int \int_set:Nn \l_@@_last_empty_row_int { #1 } \int_step_inline:nnnn { #1 } { #3 } { #5 } { #5 }
{ \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } } \bool_lazy_or:nnTF \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } } \l_tmpb_bool { \bool_set_true:N \l_tmpa_bool } { \bool_if:NF \l_tmpa_bool { \int_set:Nn \l_@@_last_empty_row_int { ##1 } } }

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

\bool_set_false:N \l_tmpa_bool \int_zero_new:N \l_@@_last_empty_column_int \int_set:Nn \l_@@_last_empty_column_int { #2 } \int_step_inline:nnnn { #2 } { #4 } { #6 } { #6 }
{ \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { #1 } } \bool_lazy_or:nnTF \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - #1 } { \bool_set_true:N \l_tmpa_bool } { \bool_if:NF \l_tmpa_bool { \int_set:Nn \l_@@_last_empty_column_int { #1 } } }

Now, we loop over the rows.

\int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int

We treat the row number ##1 with another loop.

\bool_set_false:N \l_tmpa_bool
\int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int

\@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
\bool_lazy_or:nnTF \l_tmpb_bool { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 } { \bool_set_true:N \l_tmpa_bool }
\bool_if:NF \l_tmpa_bool { \int_set:Nn \l_@@_last_empty_column_int { ####1 } \seq_put_right:Nn \l_@@_empty_corner_cells_seq { ##1 - ####1 } }

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-#2} is in a block (or in a cell with a \texttt{\textbackslash diagbox}).

\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

\cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6

\int_compare:nNnT { #3 } < { \@@_succ:n { #1 } } \int_compare:nNnT { #1 } < { \@@_succ:n { #2 } }
\int_compare:nNnT { #4 } < { \@@_succ:n { #5 } } \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
{ \bool_set_true:N \l_tmpb_bool }

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{col} nodes and the \texttt{row} nodes.
**Horizontal dotted lines**

The following command must *not* be protected because it’s meant to be expanded in a `\noalign`.

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

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

```latex
\cs_new_protected:Npn \@@_hdottedline_i:
\begin{Verbatim}
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\{ \@@_hdottedline:n { \int_use:N \c@iRow } \}
\end{Verbatim}
```

The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

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

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```latex
\cs_new_protected:Npn \@@_hdottedline_i:n \ #1
\begin{Verbatim}
\pgfrememberpicturepositiononpagetrue
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\{ \@@_hdottedline:n \{ \int_use:N \c@iRow \} \}
\end{Verbatim}
```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```latex
\cs_new_protected:Npx \@@_hdottedline:n #1
\begin{Verbatim}
\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{Verbatim}
```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```latex
\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
\dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

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{NiceMatrix}
  1 & 2 & 3 & 4 \\
  \hline
  1 & 2 & 3 & 4 \\
  \hdottedline
  1 & 2 & 3 & 4
\end{NiceMatrix}
```

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

151
\begin{bNiceMatrix}
\hline
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

\begin{pmatrix}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4
\end{pmatrix}

\begin{verbatim}
\@@_qpoint:n { col - 1 }
\dim_set:Nn \l_@@_x_initial_dim
\{
\pgf@x +
\}

We do a reduction by $\text{arraycolsep}$ for the environments with delimiters (and not for the other).
\begin{verbatim}
\bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
- \l_@@_left_margin_dim
\}
\@@_qpoint:n { col - \@@_succ:n \c@jCol }
\dim_set:Nn \l_@@_x_final_dim
\{
\pgf@x -
\bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
+ \l_@@_right_margin_dim
\}
\end{verbatim}

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \ \l_@@_inter_dots_dim$ is \textit{ad hoc} for a better result.
\begin{verbatim}
\tl_if_eq:NnF \l_@@_left_delim_tl \ ( \
\{ \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } \}
\tl_if_eq:NnF \l_@@_right_delim_tl \ ) \
\{ \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } \}
\end{verbatim}

As for now, we have no option to control the style of the lines drawn by $\hdottedline$ and the specifier \textquote{\textcolon} in the preamble. That’s why we impose the style \texttt{standard}.
\begin{verbatim}
\tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
\@@_draw_line:
\end{verbatim}

Vertical dotted lines
\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}

152
The command \texttt{\CT@arc} is a command of \texttt{colortbl} which sets the color of the rules in the array. The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded.

We do a translation par \texttt{-\l_@@_radius_dim} because we want the dotted line to have exactly the same position as a vertical rule drawn by \texttt{"|"} (considering the rule having a width equal to the diameter of the dots).

We arbitrary decrease the height of the dotted line by a quantity equal to \texttt{\l_@@_inter_dots_dim} in order to improve the visual impact.

As for now, we have no option to control the style of the lines drawn by \texttt{\hdottedline} and the specifier \texttt{"-"} in the preamble. That's why we impose the style \texttt{standard}.

The environment \texttt{\{NiceMatrixBlock\}}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

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

\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
At the end of the environment \texttt{\{NiceMatrixBlock\}}, we write in the main \texttt{.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 \texttt{\l@@first_env_block_int}).

\begin{verbatim}
\bool_if:NT \l@@block_auto_columns_width_bool
{ \iow_shipout:Nn \@mainaux \ExplSyntaxOn
  \iow_shipout:Nx \@mainaux
  { \cs_gset:cpn { @@_max_cell_width \int_use:N \g@@NiceMatrixBlock_int }
    \dim_eval:n { \g@@max_cell_width \int_use:N + \arrayrulewidth } }
  \iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@_create_extra_nodes:
{ \bool_if:nTF \l@@medium_nodes_bool
  { \bool_if:NTF \l@@large_nodes_bool \@@_create_medium_and_large_nodes:
    \@@_create_medium_nodes:
  }
  { \bool_if:NT \l@@large_nodes_bool \@@_create_large_nodes: }
}
\end{verbatim}

We have three macros of creation of nodes: \texttt{\@@create_medium_nodes:}, \texttt{\@@create_large_nodes:} and \texttt{\@@create_medium_and_large_nodes:}.

The command \texttt{\@@computations_for_medium_nodes:} must be used in a \texttt{\{pgfpicture\}}.

The extra nodes

First, two variants of the functions \texttt{\dim_min:nn} and \texttt{\dim_max:nn}.

\begin{verbatim}
\cs_generate_variant:Nn \dim_min:nn { v n }
\cs_generate_variant:Nn \dim_max:nn { v n }
\end{verbatim}

The following command is called in \texttt{\@@use_arraybox_with_notes_c}: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks dans that construction uses the standard medium nodes).

\begin{verbatim}
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
{ \cs_new_protected:Npn \@@_create_extra_nodes:
}
\end{verbatim}

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 \texttt{\@@computations_for_medium_nodes:} to do these computations.

The command \texttt{\@@computations_for_medium_nodes:} must be used in a \texttt{\{pgfpicture\}}.

For each row \texttt{i}, we compute two dimensions \texttt{l@@row_i_min_dim} and \texttt{l@@row_i_max_dim}. The dimension \texttt{l@@row_i_min_dim} is the minimal \texttt{y}-value of all the cells of the row \texttt{i}. The dimension \texttt{l@@row_i_max_dim} is the maximal \texttt{y}-value of all the cells of the row \texttt{i}. Similarly, for each column \texttt{j}, we compute two dimensions \texttt{l@@column_j_min_dim} and \texttt{l@@column_j_max_dim}. The dimension \texttt{l@@column_j_min_dim} is the minimal \texttt{x}-value of all the cells of the column \texttt{j}. The dimension \texttt{l@@column_j_max_dim} is the maximal \texttt{x}-value of all the cells of the column \texttt{j}.

Since these dimensions will be computed as maximum or minimum, we initialize them to \texttt{\c_max_dim} or \texttt{\c_min_dim}.

\begin{verbatim}
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
{ }
\end{verbatim}
We begin the two nested loops over the rows and the columns of the array.

If the cell \((i-j)\) is empty or an implicit cell (that is to say a cell after implicit ampersands &) we don’t update the dimensions we want to compute.

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

We retrieve the coordinates of the anchor north east of the (normal) node of the cell \((i-j)\). They will be stored in \texttt{\pgf@x} and \texttt{\pgf@y}.

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_col_int \g_@@_col_total_int \@@_j:
\dim_compare:nNnT
\dim_use:c { \l_@@_column_ \@@_j: \_ \min \_ \dim } = \c_max_dim
\\quad \{ \@@_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:
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\footnote{If we want to create both, we have to use \@@_create_medium_and_large_nodes:}. 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:
\tl_set:Nn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture
\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { - medium }
\@@_create_nodes:
\tl_set:Nn \l_@@_suffix_tl { - large }
\@@_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:
\tl_set:Nn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture

\footnote{If we want to create both, we have to use \@@_create_medium_and_large_nodes:}
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:
{\int_step_variable:nNn { \c@jCol - 1 } \@@_j:
  \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_{\@@_i:} _max _dim } { \l_@@_row_{\@@_i:} _min _dim }
\dim_set_eq:cc { \l_@@_column_{\@@_j:} _min _dim } { \l_@@_column_{\@@_j:} _max _dim }
\dim_set_eq:cc { \l_@@_column_{\@@_j:} _max _dim } { \l_@@_column_{\@@_j:} _max _dim }
}

\int_step_variable:nnNn { \c@iRow - 1 } \@@_i:
{\int_step_variable:nnNn { \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_{\@@_j:} _min _dim } { \l_@@_column_{\@@_j:} _max _dim }
\dim_set_eq:cc { \l_@@_column_{\@@_j:} _max _dim } { \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_@@_left_margin_dim
\dim_add:cn { \l_@@_column_{\int_use:N \c@jCol} _max _dim }
\l_@@_right_margin_dim
}

The command \l_@@_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).

\cs_new_protected:Npn \@@_create_nodes:
{\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:
}

We draw the rectangular node for the cell (\@@_i-\@@_j).

\@@_pgf_rect_node:nnnnn
{\@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
{\dim_use:c { \l_@@_column_{\@@_j:} _min _dim } }
Now, we create the nodes for the cells of the \multicolumn. We recall that we have stored in \g_@@_multicolumn_cells_seq the list of the cells where a \multicolumn{$n$}{...}{...} with $n>1$ was issued and in \g_@@_multicolumn_sizes_seq the correspondant values of $n$.

The command \@@_node_for_multicolumn:nn takes two arguments. The first is the position of the cell where the command \multicolumn{$n$}{...}{...} was issued in the format \texttt{i-j} and the second is the value of $n$ (the length of the “multi-cell”).

The blocks

The code deals with the command \Block. This command has no direct link with the environment \{NiceMatrixBlock\}.

The options of the command \Block will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

\keys_define:nn { NiceMatrix / Block / FirstPass } { \l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l , \l .value_forbidden:n = true , \r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r , \r .value_forbidden:n = true ,}
The following command \@@_Block: will be linked to Block in the environments of \nicematrix. We define it with \NewExpandableDocumentCommand because it has an optional argument between < and > (for \TeX{} instructions put before the math mode of the label and before the beginning of the small array of the block). It’s mandatory to use an expandable command.

\NewExpandableDocumentCommand \@@_Block: { O { } m D < > { } m }
{ If the first mandatory argument of the command (which is the size of the block with the syntax i-j) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).
\tl_if_blank:nTF { #2 } { \@@_Block_i 1-1 \q_stop } { \@@_Block_i #2 \q_stop } { #1 } { #3 } { #4 }
}

With the following construction, we extract the values of i and j in the first mandatory argument of the command.
\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
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 the beginning of the small array of the block and #5 is the label of the block.
\cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
{ We recall that #1 and #2 have been extracted from the first mandatory argument of \Block (which is of the syntax i-j). However, the user is allowed to omit i or j (or both). We detect that situation by replacing a missing value by 100 (it’s a convention: when the block will actually be drawn these values will be detected and interpreted as maximal possible value according to the actual size of the array).
\bool_lazy_or:nnTF
{ \tl_if_blank_p:n { #1 } }
{ \str_if_eq_p:nn { #1 } { * } }
{ \int_set:Nn \l_tmpa_int { 100 } }
{ \int_set:Nn \l_tmpa_int { #1 } }
\bool_lazy_or:nnTF
{ \tl_if_blank_p:n { #2 } }
{ \str_if_eq_p:nn { #2 } { * } }
{ \int_set:Nn \l_tmpb_int { 100 } }
{ \int_set:Nn \l_tmpb_int { #2 } }
\int_compare:nNnTF \l_tmpb_int = 1
{ \tl_if_empty:NTF \l_@@_cell_type_tl
{ \tl_set:Nn \l_@@_pos_of_block_tl c }
{ \tl_set_eq:NN \l_@@_pos_of_block_tl \l_@@_cell_type_tl }
}
{ \tl_set:Nn \l_@@_pos_of_block_tl c }
\keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
\tl_set:Nx \l_tmpa_tl
{ \int_use:N \c@iRow }
{ \int_use:N \c@jCol }
{ \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
{ \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
Now, \texttt{\l_tmpa_tl} contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets: \{imin\}-\{jmin\}-\{imax\}-\{jmax\}.

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: \texttt{\@\_Block\_iv:nnnnn} and \texttt{\@\_Block\_v:nnnnn} (the five arguments of those macros are provided by curryfication).

\begin{verbatim}
\bool_lazy_or:nnTF
{ \int_compare_p:nNn { \l_tmpa_int } = 1 }
{ \int_compare_p:nNn { \l_tmpb_int } = 1 }
{ \exp_args:Nxx \@\_Block\_iv:nnnnn }
{ \exp_args:Nxx \@\_Block\_v:nnnnn }
{ \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
\end{verbatim}

The following macro is for the case of a \texttt{Block} which is mono-row or mono-column (or both). In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array after the construction of the array (by using \texttt{pgf}).

\begin{verbatim}
\cs_new_protected:Npn \@\_Block\_iv:nnnnn #1 #2 #3 #4 #5
{ \int_gincr:N \g_@@_block_box_int
\cs_set_protected_nopar:Npn \diagbox ##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_eval:n { \c@iRow + #1 - 1 } }
    { \int_eval:n { \c@jCol + #2 - 1 } }
    { \exp_not:n { ##1 } } { \exp_not:n { ##2 } } }
  \box_gclear_new:c
  { g_@@_block_box_int_box }
  \hbox_gset:cn
  { g_@@_block_box_int_box }
}
\end{verbatim}

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with \texttt{\set@color} and not \texttt{\color\_ensure\_current}: because that command seems to be bugged: it doesn’t work in XeLaTeX when fontspec is loaded.

\begin{verbatim}
\tl_if_empty:NTF \l_@@_color_tl
{ \int_compare:nNnT { #2 } = 1 \set@color }
{ \color { \l_@@_color_tl } }
\group_begin:
\cs_set:Npn \arraystretch { 1 }
\dim_set_eq:NN \extrarowheight \c_zero_dim
\end{verbatim}

If the box is rotated (the key \texttt{\rotate} may be in the previous \texttt{#4}), the tabular used for the content of the cell will be constructed with a format \texttt{c}. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

\begin{verbatim}
\bool_if:NT \g_@@_rotate_bool
{ \tl_set:Nn \l_@@_pos_of_block_tl c }
\bool_if:NTF \l_@@_NiceTabular_bool
{ \exp_args:Nnx \begin { tabular }
{ @ { } \l_@@_pos_of_block_tl @ { } }
\end{verbatim}

160
If we are in a mono-column block, we take into account the width of that block for the width of the column.

If we are in a mono-row block, we take into account the height and the depth of that block for the height and the depth of the row.

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@pos_of_block_tl. However, maybe there were
no key of the horizontal alignment and that’s why we put a key corresponding to the value of \l_@@_pos_of_block_tl, which is fixed by the type of current column.

\{ \exp_not:n { #3 } , \l_@@_pos_of_block_tl \}
\box_use_drop:c
\{ g_@@_block _ box _ \int_use:N g_@@_block_box_int _ box \}
\}

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is not composed right now in a box. The composition in a box will be done further, just after the construction of the array.

\cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
\seq_gput_right:Nx \g_@@_blocks_seq
\l_tmpa_tl\{ \exp_not:n { #3 } \}
\exp_not:n
\{ \bool_if:NTF \l_@@_NiceTabular_bool
\group_begin:
\cs_set:Npn \arraystretch { 1 }
\dim_set_eq:NN \extrarowheight \c_zero_dim #4
\group_end:
\}
\exp_args:Nnx \begin { tabular }
\l_@@_pos_of_block_tl @ { } #5 \end { tabular }
\c_math_toggle_token
\}
\exp_args:Nnx \begin { array }
\l_@@_pos_of_block_tl @ { } #5 \end { array }
\c_math_toggle_token
\group_end:
\}

If the box is rotated (the key \rotate may be in the previous \#4), the tabular used for the content of the cell will be constructed with a format \textit{c}. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

\bool_if:NT \g_@@_rotate_bool
\{ \tl_set:Nn \l_@@_pos_of_block_tl c \}
\exp_args:Nnx \begin { tabular }
\{ \tl_set:Nn \l_@@_pos_of_block_tl @ { } \} \}
\end { tabular }
\group_end:
\}
\group_begin:
\cs_set:Npn \arraystretch { 1 }
\dim_set_eq:NN \extrarowheight \c_zero_dim #4
\bool_if:NT \g_@@_rotate_bool
\{ \tl_set:Nn \l_@@_pos_of_block_tl c \}
\c_math_toggle_token
\exp_args:Nnx \begin { array }
\{ \tl_set:Nn \l_@@_pos_of_block_tl @ { } \} \}
\end { array }
\c_math_toggle_token
\group_end:
\}

We recall that the options of the command \Block are analyzed twice: first in the cell of the array and once again when the block will be put in the array after the construction of the array (by using PGF).
\keys_define:nn { NiceMatrix / Block / SecondPass }
{
  fill .tl_set:N = \l_@@_fill_tl ,
  fill .value_required:n = true ,
  draw .tl_set:N = \l_@@_draw_tl ,
  draw .default:n = default ,
  rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
  rounded-corners .default:n = 4 pt ,
  color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
  color .value_required:n = true ,
  borders .clist_set:N = \l_@@_borders_clist ,
  borders .value_required:n = true ,
  line-width .dim_set:N = \l_@@_line_width_dim ,
  line-width .value_required:n = true ,
  l .code:n = \tl_set:Nn \l_@@_pos_of_block_tl l ,
  l .value_forbidden:n = true ,
  r .code:n = \tl_set:Nn \l_@@_pos_of_block_tl r ,
  r .value_forbidden:n = true ,
  c .code:n = \tl_set:Nn \l_@@_pos_of_block_tl c ,
  c .value_forbidden:n = true ,
  unknown .code:n = \@@_error:n { Unknown~key~for~Block }
}

\cs_new_protected:Npn \@@_draw_blocks:
{
  \cs_set_eq:NN \ialign \@@_old_ialign:
  \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
}
\cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
{
  The integer \l_@@_last_row_int will be the last row of the block and \l_@@_last_col_int its last column.
  \int_zero_new:N \l_@@_last_row_int
  \int_zero_new:N \l_@@_last_col_int

  We remind that the first mandatory argument of the command \Block is the size of the block with the special format \i-j. However, the user is allowed to omit \i or \j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in \g_@@_blocks_seq as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now.
  \int_compare:nNnTF { #3 } > { 99 }
    \{ \int_compare:nNnTF \l_@@_last_row_int <= \g_@@_static_num_of_col_int
          \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
        \group_begin:
          \globaldefs = 1
          \@@_msg_redirect_name:nn { columns-not-used } { none }
        \group_end:
    \}
}
The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also
the \multicolumn and the \diagbox in that sequence).
\seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

The group is for the keys.
\group_begin:
\keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
\tl_if_empty:NF \l_@@_draw_tl

\tl_gput_right:Nx \g_nicematrix_code_after_tl
\@@_stroke_block:nnn
{ \exp_not:n { #5 } }
{ #1 - #2 }
{ \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
\seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
{ { #1 } { #2 } { #3 } { #4 } }

\clist_if_empty:NF \l_@@_borders_clist
\tl_gput_right:Nx \g_nicematrix_code_after_tl
\@@_stroke_borders_block:nnn
{ \exp_not:n { #5 } }
{ #1 - #2 }
{ \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
\tl_if_empty:NF \l_@@_fill_tl
\tl_gput_right:Nx \g_nicematrix_code_before_tl
\exp_not:N \roundedrectanglecolor
{ \exp_not:N \l_@@_fill_tl }
{ #1 - #2 }
{ \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
{ \dim_use:N \l_@@_rounded_corners_dim }
\cs_set_protected_nopar:Npn \diagbox #1 #2
\tl_gput_right:Nx \g_@@_internal_code_after_tl
\@@_actually_diagbox:nnnnnn
{ #1 }
{ #2 }
{ \int_use:N \l_@@_last_row_int }
{ \int_use:N \l_@@_last_col_int }
Let’s consider the following \texttt{\{NiceTabular\}}. Because of the instruction \texttt{!{\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 \texttt{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 \\
& seven & eight \\
\end{NiceTabular}

We highlight the node 1-1-block

\begin{tabular}{c}
\begin{NiceTabular}{cc!{\hspace{1cm}}c}
\Block{2-2}{our block} & & one \\
& & two \\
three & four & five \\
& seven & eight \\
\end{NiceTabular}
\end{tabular}

We highlight the node 1-1-block-short

The construction of the node corresponding to the merged cells.

\begin{pgfpicture}
\@@_pgf_rect_node:nnnnn
{ \@@_env: - #1 - #2 - block }
\l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
\end{pgfpicture}

We construct the short node.

\begin{pgfpicture}
\@@_pgf_rect_node:nnnnn
\l TMPB DIM \l TMPA DIM \l TMPD DIM \l TMPC DIM
\end{pgfpicture}

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{pgfpicture}
\@@_pgf_rect_node:nnnnn
\l_00_row_total_int \l_00_first_row_int \g_00_env: - ##1 - \#2
\end{pgfpicture}

We construct the \texttt{short} node.

\begin{pgfpicture}
\@@_pgf_rect_node:nnnnn
\l_TMPB_DIM \l_TMPA_DIM \l_TMPD_DIM \l_TMPC_DIM
\end{pgfpicture}

\begin{pgfpicture}
\@@_pgf_rect_node:nnnnn
\l_00_row_total_int \l_00_first_row_int \g_00_env: - ##1 - \#2
\end{pgfpicture}

\begin{pgfpicture}
\@@_pgf_rect_node:nnnnn
\l_TMPB_DIM \l_TMPA_DIM \l_TMPD_DIM \l_TMPC_DIM
\end{pgfpicture}
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.

\dim_compare:nNnT \l_tmpb_dim = \c_max_dim
{\ @@_qpoint:n { col - #2 }
  \dim_set_eq:NN \l_tmpb_dim \pgf@x }
\dim_set:Nn \l_tmpd_dim { - \c_max_dim }
\int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
{\cs_if_exist:cT
  { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
  { \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
    { \pgfpointanchor
      { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
      { east }
    \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
  }
}
\dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
{\ @@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
  \dim_set_eq:NN \l_tmpb_dim \pgf@x }
\@@_pgf_rect_node:nnnnn
{ \@@_env: - #1 - #2 - block - short }
\l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim

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: - \int_use:N \l_@@_last_row_int
    \int_use:N \l_@@_last_col_int - 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 }
{\ @@_pgf_rect_node:nnn
  \pgfpointanchor { \@@_env: - #1 - #2 - block - medium }
  \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west }
  \pgfpointanchor
  { \@@_env:
    \int_use:N \l_@@_last_row_int
    \int_use:N \l_@@_last_col_int - medium }
  { south~east }
}

We take into account the case of a block of one row in the “first row” or the “last 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 \texttt{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 \texttt{l_tmpa_dim}.

\begin{verbatim}
\pgfextracty l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }
\end{verbatim}

We retrieve (in \texttt{pgf@x}) the x-value of the center of the block.

\begin{verbatim}
\pgfpointanchor { \@@_env: - #1 - #2 - block - short }
{ \str_case:Vn \l_@@_pos_of_block_tl
 { c { center }
  l { west }
  r { east }
 }
}
\end{verbatim}

We put the label of the block which has been composed in \texttt{l_@@_cell_box}.

\begin{verbatim}
\pgftransformshift { \pgfpoint \pgf@x l_tmpa_dim }
\pgfset { inner~sep = \c_zero_dim }
\pgfnode
{ rectangle }
{ \str_case:Vn \l_@@_pos_of_block_tl
 { c { base }
  l { base~west }
  r { base~east }
 }
}
{ \box_use_drop:N \l_@@_cell_box } { } { }
\end{verbatim}

If the number of rows is different of 1, we will put the label of the block by using the short node (the label of the block has been composed in \texttt{l_@@_cell_box}).

\begin{verbatim}
If we are in the first column, we must put the block as if it was with the key \texttt{r}.
\end{verbatim}

\begin{verbatim}
\int_compare:nNnT { #2 } = 0
{ \tl_set:Nn \l_@@_pos_of_block_tl r }
\bool_if:nT \g_@@_last_col_found_bool
{ \int_compare:nNnT { #2 } = \g_@@_col_total_int
{ \tl_set:Nn \l_@@_pos_of_block_tl l }
}
\pgftransformshift
{ \pgfpointanchor
{ \@@_env: - #1 - #2 - block - short }
{ \str_case:Vn \l_@@_pos_of_block_tl
 { c { center }
  l { west }
  r { east }
 }
}
\pgfset { inner~sep = \c_zero_dim }
\pgfnode
{ rectangle }
\str_case:Vn \l_@@_pos_of_block_tl
{ }
\end{verbatim}
The first argument of \@@_stroke_block:nnn is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax \(i-j\)) and the third is the last cell of the block (with the same syntax).

\cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
{\group_begin:
\tl_clear:N \l_@@_draw_tl
\dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
\keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\tl_if_empty:NF \l_@@_draw_tl
{If the user has used the key \color of the command \Block without value, the color fixed by \arrayrulecolor is used.
\str_if_eq:VnTF \l_@@_draw_tl { default } { \CT@arc@ }
{ \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
\pgfsetcornersarced
\@@_cut_on_hyphen:w #2 \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 }
\@@_qpoint:n { col - \l_tmpb_tl } \dim_set:Nn \l_tmpc_dim { \pgf@x }
\pgfpathrectanglecorners
\pgfpoint \l_tmpc_dim \l_tmpb_dim
\pgfpoint \l_tmpd_dim \l_tmpa_dim
\pgfusepath { stroke }
}
\pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
\endpgfpicture
\group_end:
}\endpgfpicture
\group_end:

We can’t use \pgfusepathqstroke because of the key \roundedcorners.
\pgfusepath { stroke }
Here is the set of keys for the command \@@_stroke_block:nnn.

\keys_define:nn { NiceMatrix / BlockStroke } {
  color .tl_set:N = \l_@@_draw_tl ,
  draw .tl_set:N = \l_@@_draw_tl ,
  draw .default:n = default ,
  line-width .dim_set:N = \l_@@_line_width_dim ,
  rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
  rounded-corners .default:n = 4 pt
\
}

The first argument of \@@_stroke_borders_block:nnn is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

\cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3 {\
  \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth\
  \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }\
  \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim { \@@_error:n { borders~forbidden } } {\
    \clist_map_inline:Nn \l_@@_borders_clist {\
      \clist_if_in:nnF { top , bottom , left , right } { ##1 } { \@@_error:nn { bad~border } { ##1 } }\
    }\
    \@@_cut_on_hyphen:w #2 \q_stop\
    \tl_set_eq:NN \l_tmpa_tl \l_tmpb_tl\
    \@@_cut_on_hyphen:w #3 \q_stop\
    \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }\
    \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }\
    \pgfpicture\
    \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }\
    \clist_if_in:NnT \l_@@_borders_clist { right } { \@@_stroke_vertical:n \l_tmpb_tl }\
    \clist_if_in:NnT \l_@@_borders_clist { left } { \@@_stroke_vertical:n \l_tmpd_tl }\
    \clist_if_in:NnT \l_@@_borders_clist { bottom } { \@@_stroke_horizontal:n \l_tmpa_tl }\
    \clist_if_in:NnT \l_@@_borders_clist { top } { \@@_stroke_horizontal:n \l_tmc_t }\
    \endpgfpicture\
  }\
}

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the \texttt{col} node).

\cs_new_protected:Npn \@@_stroke_vertical:n #1 {\
  \@@_qpoint:n \l_tmpc_tl\
  \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }\
  \@@_qpoint:n \l_tmpa_tl\
  \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }\
  \@@_qpoint:n \l_00\q_stop\
  \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }\
  \clist_if_in:NnT \l_@@_borders_clist { right } { \@@_stroke_vertical:n \l_tmpb_tl }\
  \clist_if_in:NnT \l_@@_borders_clist { left } { \@@_stroke_vertical:n \l_tmpd_tl }\
  \clist_if_in:NnT \l_@@_borders_clist { bottom } { \@@_stroke_horizontal:n \l_tmpa_tl }\
  \clist_if_in:NnT \l_@@_borders_clist { top } { \@@_stroke_horizontal:n \l_tmc_t }\
  \endpgfpicture\
}
The following command is used to stroke the top border and the bottom border. The argument \texttt{#1} is the number of row (in the sense of the row node).

\begin{verbatim}
\cs_new_protected:Npn \@@_stroke_horizontal:n #1 \{ \@@_qpoint:n \l_tmpd_tl \clist_if_in:NnTF \l_@@_borders_clist { left } { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } } { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } } \@@_qpoint:n \l_tmpb_tl \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim } \@@_qpoint:n { #1 } \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y } \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y } \pgfusepathqstroke \}
\end{verbatim}

Here is the set of keys for the command \texttt{\@@_stroke_borders_block:nnn}.

\begin{verbatim}
\keys_define:nn { NiceMatrix / BlockBorders } { borders .clist_set:N = \l_@@_borders_clist , rounded-corners .dim_set:N = \l_@@_rounded_corners_dim , rounded-corners .default:n = 4 pt , line-width .dim_set:N = \l_@@_line_width_dim }
\end{verbatim}

How to draw the dotted lines transparently

\begin{verbatim}
\cs_set_protected:Npn \@@_renew_matrix: \{ \RenewDocumentEnvironment { pmatrix } { } \{ \pNiceMatrix \} \{ \endpNiceMatrix \} \RenewDocumentEnvironment { vmatrix } { } \{ \vNiceMatrix \} \{ \endvNiceMatrix \} \RenewDocumentEnvironment { Vmatrix } { } \{ \VNiceMatrix \} \{ \endVNiceMatrix \} \RenewDocumentEnvironment { bmatrix } { } \{ \bNiceMatrix \} \{ \endbNiceMatrix \} \RenewDocumentEnvironment { Bmatrix } { } \{ \BNiceMatrix \} \{ \endBNiceMatrix \} \end{verbatim}

Automatic arrays

\begin{verbatim}
\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 \}
\end{verbatim}
\begin{NiceArrayWithDelims}{#1}{#2}
\{ \l_@@_nb_cols_int \, c \} \, #3, \, #5, \, #7
\int_compare:nNnT \l_@@_first_row_int = 0
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ & \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ & \} \\}
\int_compare:nNnT \l_@@_last_col_int > \{ -1 \} \{ & \} \\}
\prg_replicate:nn \l_@@_nb_rows_int
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ & \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ & \} \#6
\int_compare:nNnT \l_@@_last_col_int > \{ -1 \} \{ & \} \\}
\prg_replicate:nn \l_@@_nb_rows_int
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ & \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ & \}
\int_compare:nNnT \l_@@_last_col_int > \{ -1 \} \{ & \} \\}
\prg_replicate:nn \l_@@_nb_rows_int
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ & \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ & \}
\int_compare:nNnT \l_@@_last_col_int > \{ -1 \} \{ & \} \\}
\end{NiceArrayWithDelims}

You put \{\} before \#6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).

We define also an command \AutoNiceMatrix similar to the environment \{NiceMatrix\}.

\NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } } { 
\group_begin:
\bool_set_true:N \l_@@_NiceArray_bool
\AutoNiceMatrixWithDelims . . { #2 } { #4 } \[ #1 , #3 , #5 \]
\group_end:
}

The redefinition of the command \dotfill

First, we insert \@\dotfill (which is the saved version of \dotfill) in case of use of \dotfill “internally” in the cell (e.g. \hbox to 1cm {\dotfill}).

\bool_if:NT \l_@@_NiceTabular_bool
\AutoNiceMatrixWithDelims . . { #2 } { #4 } [ \#1 , \#3 , \#5 ]
\group_end:
}
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 \texttt{\@@_dotfill} (which is the saved version of \texttt{\dotfill}) in the cell of the array, and it will extend, since it is no longer in \texttt{l_@@_cell_box}.

\begin{verbatim}
\cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
\cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }
\end{verbatim}

The command \texttt{\diagbox}

The command \texttt{\diagbox} will be linked to \texttt{\diagbox:nn} in the environments of \texttt{nicematrix}.

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

We put the cell with \texttt{\diagbox} in the sequence \texttt{\g_@@_pos_of_blocks_seq} because a cell with \texttt{\diagbox} must be considered as non empty by the key \texttt{except-corners}.

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

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

The first four arguments of \texttt{\@@_actually_diagbox:nnnnnn} correspond to the rectangle (=block) to slash (we recall that it’s possible to use \texttt{\diagbox} in a \texttt{\Block}). The two other are the elements to draw below and above the diagonal line.

\begin{verbatim}
\pgfpathmoveto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
\pgfusepathqstroke
\end{verbatim}

The command \texttt{\CT@arc@} is a command of \texttt{colortbl} which sets the color of the rules in the array. The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded.

\begin{verbatim}
\CT@arc@
\pgfsetroundcap
\pgfusepathqstroke
\end{verbatim}
The keyword `\CodeAfter`

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs `key-value` between square brackets. Here is the corresponding set of keys.

```
\keys_define:nn { NiceMatrix }
  { 
    CodeAfter / rules .inherit:n = NiceMatrix / rules ,
    CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix 
  }
\keys_define:nn { NiceMatrix / CodeAfter }
  { 
    sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
    rules .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown-key-for-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. 97.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter`. That macro must not be protected since it begins with `\omit`.

```
\cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do not begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`.

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

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

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

```
\cs_new_protected:Npn \@@_CodeAfter_ii:n #1
  \str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } }
```

173
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.

\begin{verbatim}
\tl_gput_right:Nn \g_nicematrix_code_after_tl \{ \end \{ #1 \} \}
\@@_CodeAfter_i:n
\end{verbatim}

The delimiters in the preamble

The command \@@_delimiter:nnn will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does not concern the exterior delimiters added by \{NiceArrayWithDelims\} (and \{pNiceArray\}, \{pNiceMatrix\}, etc.).

A delimiter in the preamble of the array will write an instruction \@@_delimiter:nnn in the \g@@_internal_code_after_tl (and also potentially add instructions in the preamble provided to \array in order to add spaces between columns).

The first argument is the type of delimiter (\(,\ [\), \{ or \}). The second argument is the number of columns. The third argument is a boolean equal to \c_true_bool (resp. \c_false_true) when the delimiter must be put on the left (resp. right) side.

```lisp
\cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
\begin{verbatim}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\l@@y_initial_dim and \l@@y_final_dim will be the y-values of the extremities of the delimiter we will have to construct.
\@@qpoint:n { row - 1 }
\dim_set_eq:NN \l@@y_initial_dim \pgf@y
\@@qpoint:n { row - \@@succ:n \c@iRow }
\dim_set_eq:NN \l@@y_final_dim \pgf@y
\end{verbatim}
```

We will compute in \l_tmpa_dim the x-value where we will have to put our delimiter (on the left side or on the right side).

```lisp
\bool_if:nTF { #3 }
\begin{verbatim}
\{ \dim_set_eq:NN \l_tmtpa_dim \c_max_dim \}
\{ \dim_set:Nn \l_tmtpa_dim \c_max_dim \}
\int_step_inline:nnn \l@@first_row_int \g@@row_total_int
\cs_if_exist:cT { pgf@sh@ns@\@@env:-##1-#2 }
\pgfpointanchor { \@@env:-\#1-\#2 } { east } \dim_set:Nn \l_tmtpa_dim
\bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmtpa_dim \pgf@x
\end{verbatim}
```

Now we can put the delimiter with a node of PGF.

```lisp
\pgfset { inner-sep = \c_zero_dim }
\dim_zero:N \nulldelimiterspace
\pgftransformshift
\begin{verbatim}
\pgfpoint { \l_tmtpa_dim }
\pgfpointanchor { \l@@y_initial_dim + \l@@y_final_dim + \arrayrulewidth } / 2
\end{verbatim}
```

\end{verbatim}
Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

\nullfont
\c_math_toggle_token
\tl_if_empty:NF \l_@@_delimiters_color_tl
{ \color { \l_@@_delimiters_color_tl } }
\bool_if:nTF { #3 } { \left #1 } { \left . }
vcenter
{ }
{ }
\endpgfpicture

The command \SubMatrix
\keys_define:nn { NiceMatrix / sub-matrix }
{
  extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
  extra-height .value_required:n = true ,
  left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
  left-xshift .value_required:n = true ,
  right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
  right-xshift .value_required:n = true ,
  xshift .value_required:n = true ,
  delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
  delimiters / color .value_required:n = true ,
  slim .bool_set:N = \l_@@_submatrix_slim_bool ,
  slim .default:n = true ,
  hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
  hlines .default:n = all ,
  vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
  vlines .default:n = all ,
  hvlines .meta:n = { hlines, vlines } ,
  hvlines .value_forbidden:n = true ,
}
\keys_define:nn { NiceMatrix / SubMatrix }
{
  SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
  CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
  NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
  NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
  NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
}

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).
\keys_define:nn { NiceMatrix / SubMatrix }
{
hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
hlines .default:n = all ,
vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
vlines .default:n = all ,
hvlines .meta:n = { hlines, vlines },
hvlines .value_forbidden:n = true ,
name .code:n =
\tl_if_empty:nTF { #1 }
  { \@@_error:n { Invalid-name-format } }
  {
    \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 } { #1 }
    {
      \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
        { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
      \str_set:Nn \l_@@_submatrix_name_str { #1 }
      \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
    }
  }
  { \@@_error:n { Invalid-name-format } }
  ,
rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 },
rules .value_required:n = true ,
code .tl_set:N = \l_@@_code_tl ,
code .value_required:n = true ,
name .value_required:n = true ,
unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
)

\NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m O { } }
  { \@@_cut_on_hyphen:w #3 \q_stop
    \tl_clear_new:N \l_@@_first_i_tl
    \tl_clear_new:N \l_@@_first_j_tl
    \tl_clear_new:N \l_@@_last_i_tl
    \tl_clear_new:N \l_@@_last_j_tl
    \@@_cut_on_hyphen:w #2 \q_stop
    \seq_gput_right:Nx \g_@@_submatrix_seq
      { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
    \tl_gput_right:Nn \g_@@_internal_code_after_tl
      { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
)

In the internal code-after and in the \CodeAfter the following command \@@_SubMatrix will be linked to \SubMatrix.
  • #1 is the left delimiter;
  • #2 is the upper-left cell of the matrix with the format i-j;
  • #3 is the lower-right cell of the matrix with the format i-j;
  • #4 is the right delimiter;
  • #5 is the list of options of the command.

\NewDocumentCommand \@@_SubMatrix { m m m m O { } }
  { \group_begin:
    The four following token lists correspond to the position of the \SubMatrix.
      \tl_clear_new:N \l_@@_first_i_tl
      \tl_clear_new:N \l_@@_first_j_tl
      \tl_clear_new:N \l_@@_last_i_tl
      \tl_clear_new:N \l_@@_last_j_tl
The command \@@_cut_on_hyphen:w cuts on the hyphen an argument of the form \textit{i-j}. The value of \textit{i} is stored in \l_tmpa_tl and the value of \textit{j} is stored in \l_tmpb_tl.

\begin{verbatim}
5582 \@@_cut_on_hyphen:w #2 \q_stop
5583 \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
5584 \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
5585 \@@_cut_on_hyphen:w #3 \q_stop
5586 \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
5587 \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
5588 \bool_lazy_or:nnTF
5589 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
5590 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
5591 { \@@_error:n { SubMatrix-too-large } }
5592 {
5593 \str_clear_new:N \l_@@_submatrix_name_str
5594 \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
5595 \pgfpicture
5596 \pgfrememberpicturepositiononpagetrue
5597 \pgfset { inner-\textit{sep} = \c_zero_dim }
5598 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
5599 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
5600 \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl { \\int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int { #5 } }
5601 \cs_if_exist:cT
5602 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
5603 { \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
5604 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x }
5605 }\cs_if_exist:cT
5606 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
5607 { \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
5608 \dim_set:Nn \l_@@_x_final_dim { \pgf@x }
5609 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl { \\int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int { #5 } }
5610 \cs_if_exist:cT
5611 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
5612 { \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
5613 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y }
5614 }\cs_if_exist:cT
5615 { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
5616 { \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
5617 \dim_set:Nn \l_@@_y_final_dim { \pgf@y }
5618 }
5619 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
5620 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
5621 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
5622 \dim_set:Nn \l_@@_y_final_dim { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
5623 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int { #5 } }
5624 \cs_if_exist:cT
5625 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - \#1 }
5626 { \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - \#1 } { north }
5627 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y }
5628 }\cs_if_exist:cT
5629 { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - \#1 }
5630 { \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - \#1 } { south }
5631 \dim_set:Nn \l_@@_y_final_dim { \pgf@y }
5632 }
\end{verbatim}

The last value of \texttt{\int_step_inline:nnn} is provided by currification.

\begin{verbatim}
5633 \bool_if:NTF \l_@@_submatrix_slim_bool
5634 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl { \\int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int } }
5635 \cs_if_exist:cT
5636 { pgf @ sh @ ns @ \@@_env: - \#1 - \l_@@_first_j_tl }
5637 { \pgfpointanchor { \@@_env: - \#1 - \l_@@_first_j_tl } { west }
5638 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x }
5639 }\cs_if_exist:cT
5640 { pgf @ sh @ ns @ \@@_env: - \#1 - \l_@@_last_j_tl }
5641 { \pgfpointanchor { \@@_env: - \#1 - \l_@@_last_j_tl } { east }
5642 \dim_set:Nn \l_@@_x_final_dim { \pgf@x }
5643 }
5644 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
5645 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
5646 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
5647 \dim_set:Nn \l_@@_y_final_dim { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
5648 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int { #5 } }
5649 \cs_if_exist:cT
5650 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - \#1 }
5651 { \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - \#1 } { north }
5652 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y }
5653 }\cs_if_exist:cT
5654 { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - \#1 }
5655 { \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - \#1 } { south }
5656 \dim_set:Nn \l_@@_y_final_dim { \pgf@y }
5657 }
\end{verbatim}
We will draw the rules in the \SubMatrix.

Now, we draw the potential vertical rules specified in the preamble of the environments with the letter fixed with the key \vlines-in-sub-matrix. The list of the columns where there is such rule to draw is in \g@@_cols_vlism_seq.

Now, we draw the vertical rules specified in the key \vlines of \SubMatrix. The list of the columns where there is such rule to draw is in \g@@_cols_vlism_seq.

Now, we draw the horizontal rules specified in the key \hlines of \SubMatrix. The last argument of \int_step_inline:nn or \clist_map_inline:NN is given by curryfication.
We use a group to protect \l_tmpa_dim and \l_tmpb_dim.

\group_begin:

We compute in \l_tmpa_dim the x-value of the left end of the rule.

\dim_set:Nn \l_tmpa_dim
\{ \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim \}

\str_case:nn { #1 }
{ }
{ \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
{ \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
{ \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }

\pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

We compute in \l_tmpb_dim the x-value of the right end of the rule.

\dim_set:Nn \l_tmpb_dim
\{ \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim \}

\str_case:nn { #4 }
{ }
{ \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
{ \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
{ \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

\pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
\pgfusepathqstroke

\group_end:
\{ \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } \}

If the key name has been used for the command \SubMatrix, we create a PGF node with that name for the submatrix (this node does not encompass the delimiters that we will put after).

\str_if_empty:NF \l_@@_submatrix_name_str
{ }
\{ \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
\l_@@_x_initial_dim \l_@@_y_initial_dim
\l_@@_x_final_dim \l_@@_y_final_dim
\}

\group_end:

The group was for \CT@arc@ (the color of the rules).

Now, we deal with the left delimiter. Of course, the environment \pgfscope is for the \pgftransformshift.

\begin { pgfscope }
\pgftransformshift
{ \pgfpoint
\{ \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim \}
\{ \l_@@_y_initial_dim + \l_@@_y_final_dim / 2 \}
}

\str_if_empty:NF \l_@@_submatrix_name_str
{ }
{ \@@_node_left:nn #1 { } }
{ \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
\end { pgfscope }

Now, we deal with the right delimiter.

\pgftransformshift
{ }

179
The group was a group for the whole \SubMatrix. In the key code of the command \SubMatrix there may be Tikz instructions. We want that, in these instructions, the \textit{i} and \textit{j} in specifications of nodes of the forms \textit{i-j}, row-\textit{i}, col-\textit{j} and \textit{i}-|\textit{j} refer to the number of row and column \textit{relative} of the current \SubMatrix. That’s why we will patch (locally in the \SubMatrix) the command \pgfpointanchor.

The following command will be linked to \pgfpointanchor just before the execution of the option code of the command \SubMatrix. In this command, we catch the argument \texttt{#1} of \pgfpointanchor and we apply to it the command \texttt{\@@_pgfpointanchor_i:nn} before passing it to the original \pgfpointanchor. We have to act in an expandable way because the command \pgfpointanchor is used in names of Tikz nodes which are computed in an expandable way.

In fact, the argument of \pgfpointanchor is always of the form \texttt{\textbackslash a\_command \{ name\_of\_node \}} where “\texttt{name\_of\_node}” is the name of the Tikz node without the potential prefix and suffix. That’s why we catch two arguments and work only on the second by trying (first) to extract an hyphen \texttt{-}.

Since \texttt{\seq\_if\_in:NNTF} and \texttt{\clist\_if\_in:NNTF} are not expandable, we will use the following token list and \texttt{\str\_case:NNTF} to test whether we have an integer or not.

If there is no hyphen, that means that the node is of the form of a single number (ex.: \texttt{5} or \texttt{11}). In that case, we are in an analysis which result from a specification of node of the form \texttt{i}-|\texttt{j}. In that case, the \textit{i} of the number of row arrives first (and alone) in a \pgfpointanchor and, the, the \textit{j} arrives (alone) in the following \pgfpointanchor. In order to know whether we have a number of row of
a number of column, we keep track of the number of such treatments by the expandable flag called \nicematrix.

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row-$i$ or col-$j$.

There was an hyphen in the name of the node and that’s why we have to retrieve the extra hyphen we have put (cf. \@@_pgfpointanchor_i:nn).

Now the case of a node of the form $i-j$.

The command \@@_node_left:nn puts the left delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key name has been used in \SubMatrix).

181
The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument \#1 is the delimiter to put. The argument \#2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```
cs_new_protected:Npn \@@_node_right:nn #1 #2
  {
    \pgfnode
      { rectangle }
      { west }
      {
        \nullfont
        \c_math_toggle_token
        \tl_if_empty:NF \l_@@_delimiters_color_tl
          { \color { \l_@@_delimiters_color_tl } }
        \left .
        \vcenter
          {
            \nullfont
            \hrule \@height \l_tmpa_dim
            \@depth \c_zero_dim
            \@width \c_zero_dim
          }
        \right #1
        \c_math_toggle_token
      }
      { #2 }
      { }
  }
```

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
\@@_msg_new:nnn { Unknown~option~for~package }
  { The~key~'\l_keys_key_str'~is~unknown. \`
    If~you~go~on,~it~will~be~ignored. \`
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
\@@_msg_new:nn { Key~transparent }
  { The~key~'transparent'~is~now~obsolete~(because~it's~name~
    is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
  }
```
and 'renew-matrix'. However, you can go on.

\keys_define:nn { NiceMatrix / Package }
{
  define-L-C-R .bool_set:N = \c_@@_define_L_C_R_bool ,
  define-L-C-R .default:n = true ,
  renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
  renew-dots .value_forbidden:n = true ,
  renew-matrix .code:n = \@@_renew_matrix: ,
  renew-matrix .value_forbidden:n = true ,
  transparent .code:n =
  \@@_renew_matrix:
  \bool_set_true:N \l_@@_renew_dots_bool
  \@@_error:n { Key: transparent }
},
  transparent .value_forbidden:n = true ,
  footnote .bool_set:N = \c_@@_footnote_bool ,
  footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
  unknown .code:n = \@@_error:n { Unknown: option: for: package }
}
\ProcessKeysOptions { NiceMatrix / Package }

\@@_msg_new:nn { footnote-with-footnotehyper-package }
{
  You can't use the option 'footnote' because the package
  footnotehyper has already been loaded. -
  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. \}
  If you go on, the package footnotehyper won't be loaded.
}
\@@_msg_new:nn { footnotewith-footnotepackage }
{
  You can't use the option 'footnote' because the package
  footnotehyper has already been loaded. -
  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
{
The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.
\@ifclassloaded { beamer }
{
  \bool_set_false:N \c_@@_footnote_bool
  \@ifpackageloaded { footnotehyper }
  \@@_error:n { footnote-with-footnotehyper-package }
  \usepackage { footnote }
}
}
\bool_if:NT \c_@@_footnotehyper_bool
{
The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.
\@ifclassloaded { beamer }

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

If the user uses too much columns, the command $@@error_{too\_much\_cols}$: is executed. This command raises an error but try to give the best information to the user in the error message. The command $\seq_if_in:NVTTF$ is not expandable and that's why we can't put it in the error message itself. We have to do the test before the $@@fatal:n$.

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.

\@@_msg_new:nn { too-much-cols-for-array }
\{ 
You-try-to-use-more-columns-than-allowed-by-your-
\@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
\int_use:N \g_@@_static_num_of_col_int\ (plus-the-potential-exterior-ones)-
This-error-is-fatal.
\}
\@@_msg_new:nn { last-col-not-used }
\{ 
The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column-
in-your-\@@_full_name_env:.-However,-you-can-go-on.
\}
\@@_msg_new:nn { columns-not-used }
\{ 
The-preamble-of-your-\@@_full_name_env: \ announce-\int_use:N \g_@@_static_num_of_col_int\ columns-but-you-use-only-\int_use:N \c@jCol.\ \You-can-go-on-but-the-columns-you-did-not-used-won’t-be-created.
\}
\@@_msg_new:nn { in-first-col }
\{ 
You-can’t-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\ \If-you-go-on,-this-command-will-be-ignored.
\}
\@@_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-key-will-be-ignored.
\}
\@@_msg_new:nn { Unknown-key-for-xdots }
\{ 
As-for-now,-there-is-only-three-key-available-here:-'color',-'line-style'-and-'shorten'-(and-you-try-to-use-‘\l_keys_key_str’).-If-you-go-on,-this-key-will-be-ignored.
\}
\@@_msg_new:nn { Unknown-key-for-rowcolors }
\{ 
As-for-now,-there-is-only-two-keys-available-here:-'cols'-and-'respect-blocks'-(and-you-try-to-use-‘\l_keys_key_str’).-If-you-go-on,-this-key-will-be-ignored.
\}
You can't use an ampersand (%token_to_str:N &) to separate columns because you have used the key 'light-syntax'. This error is fatal.

Your command \SubMatrix\ can't be drawn because your matrix is too small. If you go on, this command will be ignored.

You can't use %token_to_str:N \ to separate rows because you have used the key 'light-syntax'. You must use the character '\l_@@_end_of_row_tl' (set by the key 'end-of-row'). This error is fatal.

The key 'standard-cline' is available only in the preamble. If you go on, this command will be ignored.

The column type 'l' is no longer defined in 'nicematrix'. Since version 5.0, you have to use 'l', 'c' and 'r' instead of 'L', 'C' and 'R'. You can also use the key 'define-L-C-R'. This error is fatal.

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.

You can't give the name '\l_keys_value_tl' to a \SubMatrix. A name must be accepted by the regular expression [A-Za-z][A-Za-z0-9]*. If you go on, this key will be ignored.
If you go on, this command will be ignored.}
\@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
{ The-name-#1-is-already-used-for-a-\token_to_str:N \SubMatrix\ in-this-\@@_full_name_env:\ \ If-you-go-on,-this-key-will-be-ignored.\}
For-a-list-of-the-names-already-used,-</token_to_str:N >
\@@_msg_new:nnn { r-or-l-with-preamble }
{ You-can't-use-the-key-'\l_keys_key_str'-in-your-\@@_full_name_env:.- You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
your-\@@_full_name_env:\ \ If-you-go-on,-this-key-will-be-ignored. }
\@@_msg_new:nnn { Hdotsfor-in-col-0 } { You-can't-use-\token_to_str:N \Hdotsfor\ in-an-exterior-column-of-the-array.-This-error-is-fatal. }
\@@_msg_new:nnn { 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:nnn { bad-border } { #1-is-an-incorrect-specification-for-a-border-(in-the-key-
'borders'-of-the-command-\token_to_str:N \Block).-The-available-
values-are:left,-right,-top-and-bottom.\ \ If-you-go-on,-this-specification-of-border-will-be-ignored. }
\@@_msg_new:nnn { last-col-non-empty-for-NiceArray } { In-the-\@@_full_name_env:,you-must-use-the-key-
'last-col'-without-value.\ \ However,-you-can-go-on-for-this-time-
(the-value-''l_keys_value_tl''-will-be-ignored). }
\@@_msg_new:nnn { last-col-non-empty-for-NiceMatrixOptions } { In-NiceMatrixOptions,you-must-use-the-key-
'last-col'-without-value.\ \ However,-you-can-go-on-for-this-time-
(the-value-''l_keys_value_tl''-will-be-ignored). }
\@@_msg_new:nnn { Block-too-large-1 } { You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is-
too-small-for-that-block. \ }
The preamble of your \@@_full_name_env: announces \int_use:N \g_@@_static_num_of_col_int:
columns but you use only \int_use:N \c@jCol: and that's why a block specified in the cell #1–#2 can't be drawn. You should add some ampersands (&) at the end of the first row of your \@@_full_name_env:.
If you go on, this block and maybe others will be ignored.

The column type '#1' in your \@@_full_name_env: is unknown. This error is fatal.

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.

You can't use the key 'borders' of the command \token_to_str:N \Block: because the option 'rounded-corners' is in force with a non-zero value.
If you go on, this key will be ignored.

You can't use the command \token_to_str:N \tabularnote: because you haven't loaded 'booktabs'.
If you go on, this key will be ignored.

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.

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).

Environments of \nicematrix: can't be nested. This error is fatal.

The \@@_full_name_env: can be used only in math mode (and not in \token_to_str:N \vcenter:).
This error is fatal.
The value of key \l_keys_key_str 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_str 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 keys are (in alphabetic order): borders, c, draw, fill, l, line-width, rounded-corners and r. \}

\@@_msg_new:nnn { Unknown key for CodeAfter }
\{ The key \l_keys_key_str is unknown. If you go on, it will be ignored. For a list of the available keys in \token_to_str:N \CodeAfter, type H<return>. \}
\{ The available keys are (in alphabetic order): delimiters/color, rules (with the subkeys 'color' and 'width'), sub-matrix (several subkeys) and xdots (several subkeys). The latter is for the command \token_to_str:N \line. \}

\@@_msg_new:nnn { Unknown key for SubMatrix }
\{ The key \l_keys_key_str is unknown. If you go on, this key will be ignored. For a list of the available keys in \token_to_str:N \SubMatrix, type H<return>. \}
\{ The available keys are (in alphabetic order): 'delimiters/color', 'extra-height', 'hlines', 'hvlines', 'left-xshift', 'name', 'right-xshift', 'rules' (with the subkeys 'color' and 'width'), 'slim', 'vlines' and 'xshift' (which sets both 'left-xshift' and 'right-xshift'). \}

\@@_msg_new:nnn { Unknown key for notes }
\{ The key \l_keys_key_str is unknown. If you go on, it will be ignored. For a list of the available keys about notes, type H<return>. \}
\{ The available keys are (in alphabetic order): bottomrule, code-after, code-before, enumitem-keys, enumitem-keys-para, \}

189
\@@_msg_new:nnn { Unknown-key-for-NiceMatrixOptions }
{ The-key-"\l_keys_key_str"-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-keys-are-(in-alphabetic-order):- allow-duplicate-names,- cell-space-bottom-limit,- cell-space-limits,- 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,- delimiters/color,- end-of-row,- first-col,- first-row,- 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,- rules-(with-the-subkeys-"color"-and-"width"),- small,- sub-matrix-(several-subkeys), vlines,- xdots-(several-subkeys). }
\@@_msg_new:nnn { Unknown-option-for-NiceArray }
{ The-key-"\l_keys_key_str"-is-unknown-for-the-environment \{NiceArray\}. }\ 
If-you-go-on,-it-will-be-ignored. \ 
For-a-list-of-the-*principal*-available-keys,-type-H<return>. }
{ The-available-keys-are-(in-alphabetic-order):- b,- baseline,- c,- cell-space-bottom-limit,- cell-space-limits,- cell-space-top-limit,-
This error message is used for the set of keys \texttt{NiceMatrix/NiceMatrix} and \texttt{NiceMatrix/pNiceArray} (but not \texttt{NiceMatrix/NiceArray} because, for this set of keys, there is also the keys \texttt{t}, \texttt{c} and \texttt{b}).

\begin{verbatim}
\@_msg_new:nnn { Unknown-option-for-NiceMatrix }
\{
  The-key-"\l_keys_key_str"-is-unknown-for-the-
  \@_full_name_env:. \"
  If-you-go-on,-it-will-be-ignored. \"
  For-a-list-of-the-*principal*-available-keys,-type-H<return>.
}\}
\{
  The-available-keys-are-(in-alphabetic-order):-
  b,-
  baseline,-
  c,-
  cell-space-bottom-limit,-
  cell-space-limits,-
  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,-
  delimiters/color,-
  extra-left-margin,-
  extra-right-margin,-
  first-col,-
  first-row,-
  hlines,-
  hlines,-
  hlines-except-corners,-
  last-col,-
  last-row,-
  left-margin,-
  light-syntax,-
  name,-
  notes/bottomrule,-
  notes/para,-
  nullify-dots,-
  renew-dots,-
  right-margin,-
  rules-(with-the-subkeys-'color'-and-'width'),-
  small,-
  t,-
  vlines,-
  xdots/color,-
  xdots/shorten-and-
  xdots/line-style.
\}
\end{verbatim}
<table>
<thead>
<tr>
<th>delimiters/color,-</th>
<th>extra-left-margin,-</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra-right-margin,-</td>
<td>first-col,-</td>
</tr>
<tr>
<td>first-row,-</td>
<td>hlines,-</td>
</tr>
<tr>
<td>hvlines,-</td>
<td>hvlines-except-corners,-</td>
</tr>
<tr>
<td>l,-</td>
<td>last-col,-</td>
</tr>
<tr>
<td>last-row,-</td>
<td>left-margin,-</td>
</tr>
<tr>
<td>light-syntax,-</td>
<td>name,-</td>
</tr>
<tr>
<td>nullify-dots,-</td>
<td>r,-</td>
</tr>
<tr>
<td>renew-dots,-</td>
<td>right-margin,-</td>
</tr>
<tr>
<td>rules-(with-the-subkeys-'color'-and-'width'),-</td>
<td>small,-</td>
</tr>
<tr>
<td>t,-</td>
<td>vlines,-</td>
</tr>
<tr>
<td>xdots/color,-</td>
<td>xdots/shorten-and-</td>
</tr>
<tr>
<td>xdots/line-style.</td>
<td></td>
</tr>
</tbody>
</table>

}\@_msg_new:nnn \{ Unknown-option-for-NiceTabular \}
{ \{NiceTabular\}. } \{{ The-key-\"l_keys_key_str\"-is-unknown-for-the-environment- \{NiceTabular\}. } \{ If-you-go-on,-it-will-be-ignored. \} \{ For-a-list-of-the-*principal*-available-keys,-type-H<return>. \}
{ The-available-keys-are-(in-alphabetic-order):- b,- baseline,- c,- cell-space-bottom-limit,- cell-space-limits,- 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,- first-row,- hlines,- hvlines,- hvlines-except-corners,- last-col,- last-row,- left-margin,- light-syntax,- name,- |
18 History

The successive versions of the file \texttt{nicematrix.sty} provided by \TeX\ Live are available on the \texttt{svn} server of \TeX\ Live:

\url{https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty}

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

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

Changes between version 1.2 and 1.3

New environment \texttt{\{pNiceArrayC\}} and its variants.
Correction of a bug in the definition of \texttt{\{BNiceMatrix\}}, \texttt{\{vNiceMatrix\}} and \texttt{\{VNiceMatrix\}} (in fact, it was a typo).
Options are now available locally in \texttt{\{pNiceMatrix\}} and its variants.
The names of the options are changed. The old names were names in “camel style”.

193
Changes between version 1.3 and 1.4

The column types \( w \) and \( W \) 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 \texttt{nicematrix} were focused on the continuous dotted lines whereas the version 2.0 of \texttt{nicematrix} provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment \{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\footnote{\url{tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}}, Tikz externalization is now deactivated in the environments of the package \texttt{nicematrix}.

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like \texttt{\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}).

\[
\begin{pmatrix}
C_j \\
\vdots & 0 \\
0 & \ddots \\
0 & 0 \\
\end{pmatrix}
L_i
\]

Changes between version 2.1.3 and 2.1.4

Replacement of some options \texttt{0} \{ \texttt{1} \} in commands and environments defined with \texttt{xparse} by \texttt{! 0} \{ \} (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}.

\footnote{\url{tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}}
\footnote{Before this version, there was an error when using \texttt{nicematrix} with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by \texttt{nicematrix} because they use the options \texttt{overlay} and \texttt{remember picture}.}
Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`). Possibility to draw vertical dotted lines to separate columns with the specifier `;` in the preamble (similar to the classical specifier `|` and the specifier `;` of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier `;` in the preamble. Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`. Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `vlines` (of `|`) as `\Hdotsfor` does. Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices. Error message when the user gives an incorrect value for `last-row`. A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\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 `exp3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange\textsuperscript{59}, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

\textsuperscript{59}cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize
Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the \texttt{code-after} was not executed.

Changes between version 3.5 and 3.6

LaTeX counters \texttt{iRow} and \texttt{jCol} available in the cells of the array.

Addition of \texttt{\normalbaselines} before the construction of the array: in environments like \texttt{\{align\}} of \texttt{amsmath} the value of \texttt{\baselineskip} is changed and if the options \texttt{first-row} and \texttt{last-row} were used in an environment of \texttt{nicematrix}, the position of the delimiters was wrong.

A warning is written in the \texttt{.log} file if an obsolete environment is used.

There is no longer artificial errors \texttt{Duplicate-name} in the environments of \texttt{amsmath}.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: \texttt{code-for-first-col}, \texttt{code-for-last-col}, \texttt{code-for-first-row} and \texttt{code-for-last-row}.

New command \texttt{\pAutoNiceMatrix} and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command \texttt{\Block} when the block has only one row. With this programmation, the vertical rules drawn by the specifier \texttt{“|”} at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with \texttt{\multicolumn}.

An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands \texttt{\NiceMatrixLastEnv} and \texttt{\OnlyMainNiceMatrix}.

New options \texttt{create-medium-nodes} and \texttt{create-large-nodes}.

Changes between version 3.9 and 3.10

New option \texttt{light-syntax} (and \texttt{end-of-row}).

New option \texttt{dotted-lines-margin} for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to \texttt{first-row} and \texttt{last-row}.

Changes between versions 3.11 and 3.12

Command \texttt{\rotate} in the cells of the array.

Options \texttt{vlines}, \texttt{hlines} and \texttt{hvlines}.

Option \texttt{baseline} pour \texttt{\{NiceArray\}} (not for the other environments).

The name of the Tikz nodes created by the command \texttt{\Block} has changed: when the command has been issued in the cell \texttt{i-j}, the name is \texttt{i-j-block} and, if the creation of the “medium nodes” is required, a node \texttt{i-j-block-medium} is created.

If the user tries to use more columns than allowed by its environment, an error is raised by \texttt{nicematrix} (instead of a low-level error).

The package must be loaded with the option \texttt{obsolete-environments} if we want to use the deprecated environments.
Changes between versions 3.12 and 3.13

The behaviour of the command \texttt{\rotate} is improved when used in the “last row”. The option \texttt{dotted-lines-margin} has been renamed in \texttt{xdots/shorten} and the options \texttt{xdots/color} and \texttt{xdots/line-style} have been added for a complete customisation of the dotted lines.

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

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

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

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on \texttt{stackoverflow}).

Better error messages when the user uses & or \texttt{\textbackslash} when \texttt{light-syntax} is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by \texttt{\Ldots}, \texttt{\Cdots}, \texttt{\Vdots}, \texttt{\Ddots}, \texttt{\Iddots}, \texttt{\Hdots} for and the command \texttt{\line} in the \texttt{code-after} with the tokens _ and ^.

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

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

Changes between versions 3.15 and 4.0

New environment \{\texttt{NiceTabular} \}

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys \texttt{cell-space-top-limit} and \texttt{cell-space-bottom-limit}
New command \texttt{\diagbox}
The key \texttt{hvline} don’t draw rules in the blocks (commands \texttt{\Block}) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write \texttt{\begin{pNiceMatrix} a \& b \\textbackslash 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 \texttt{\Block} is correct even when an instruction such as \texttt{!{\quad}} is used in the preamble of the array.

It’s now possible to use the command \texttt{\Block} in the “last row”.

Changes between versions 4.3 and 4.4

New key \texttt{hvlines-except-corners}.
Changes between versions 4.4 and 5.0

Use of the standard column types 1, 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*}
Command 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 may take in as value an expression of the form line-i to align the \hline in the row i.
The key hvlines-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 l 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).

Changes between versions 5.3 and 5.4

Key tabularnote.
Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put \omit before \CodeAfter.
Correction of a bug: the tabular notes \tabularnotes were not composed when present in a block
(except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.
New command \NotEmpty.

Changes between versions 5.6 and 5.7

New key delimiters-color
Keys fill, draw and line-width for the command \Block.

Changes between versions 5.7 and 5.8

Keys cols and restart of the command \rowcolors in the code-before.
Modification of the behaviour of \\ in the columns of type p, m or b (for a behaviour similar to the
environments of array).
Better error messages for the command \Block.
Changes between versions 5.8 and 5.9
Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10
New command \texttt{\SubMatrix} available in the \texttt{\CodeAfter}.
It's possible to provide options (between brackets) to the keyword \texttt{\CodeAfter}.
A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11
It's now possible, in the \texttt{code-before} and in the \texttt{\CodeAfter}, to use the syntax \{(i-|j)\} for the Tikz node at the intersection of the (potential) horizontal rule number $i$ and the (potential) vertical rule number $j$.

Changes between versions 5.11 and 5.12
Keywords \texttt{\CodeBefore} and \texttt{\Body} (alternative syntax to the key `code-before`).
New key `delimiters/max-width`.
New keys `hlines`, `vlines` and `hvlines` for the command \texttt{\SubMatrix} in the \texttt{\CodeAfter}.
New key `rounded-corners` for the command \texttt{\Block}.

Changes between versions 5.12 and 5.13
New command \texttt{\arraycolor} in the \texttt{\CodeBefore} (with its key `except-corners`).
New key `borders` for the command \texttt{\Block}.
New command \texttt{\Hline} (for horizontal rules not drawn in the blocks).
The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

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>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>\CodeBefore</td>
<td>\CodeAfter_keys: \ldots 1232, 2422, 2444</td>
</tr>
<tr>
<td>_Block:</td>
<td>\CodeAfter</td>
</tr>
<tr>
<td>_Block_i:</td>
<td>\Ddots</td>
</tr>
<tr>
<td>_Block_ii:</td>
<td>_Ddots_lines_tl</td>
</tr>
<tr>
<td>_Block_iv:</td>
<td>_Hdots_for_lines_tl</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hdots_for_n_n</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hdots_for_i</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hline_i</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hline_i_n</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hline_ii_n</td>
</tr>
<tr>
<td>_Block_v:</td>
<td>_Hline_iii_n</td>
</tr>
<tr>
<td>_Cell:</td>
<td>_Hspace:</td>
</tr>
<tr>
<td>_Cell:</td>
<td>_Iddots</td>
</tr>
<tr>
<td>_Cell:</td>
<td>_Iddots_lines_tl</td>
</tr>
<tr>
<td>_Cell:</td>
<td>_Ldots</td>
</tr>
<tr>
<td>_Cell:</td>
<td>_Ldots_lines_tl</td>
</tr>
<tr>
<td>_CodeAfter_i_i_n</td>
<td>\l_Matrix_bool</td>
</tr>
</tbody>
</table>
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
   4.1 General case ........................................................................ 4
   4.2 The mono-column blocks ...................................................... 5
   4.3 The mono-row blocks .......................................................... 6
   4.4 The mono-cell blocks .......................................................... 6
   4.5 A small remark ..................................................................... 6
5 The rules
   5.1 Some differences with the classical environments ..................... 7
       5.1.1 The vertical rules ......................................................... 7
       5.1.2 The command \cline ...................................................... 8
   5.2 The thickness and the color of the rules .................................... 8
   5.3 The command \Hline ............................................................. 8
   5.4 The keys hlines and vlines .................................................... 9
   5.5 The key hvlines .................................................................... 9
   5.6 The key hvlines-except-corners ............................................ 9
   5.7 The command \diagbox ........................................................ 10
   5.8 Dotted rules ......................................................................... 10
6 The color of the rows and columns .............................................. 11
   6.1 Use of colortbl ..................................................................... 11
   6.2 The tools of nicematrix in the \CodeBefore ................................. 11
   6.3 Color tools with the syntax of colortbl ..................................... 15
7 The width of the columns .......................................................... 16
8 The exterior rows and columns .................................................... 17
9 The continuous dotted lines ...................................................... 18
   9.1 The option nullify-dots ....................................................... 19
   9.2 The commands \Hdotsfor and \Vdotsfor .................................... 20
   9.3 How to generate the continuous dotted lines transparently ............ 21
   9.4 The labels of the dotted lines ............................................... 21
   9.5 Customisation of the dotted lines .......................................... 22
   9.6 The dotted lines and the rules .............................................. 23
10 The \CodeAfter ........................................................................ 23
   10.1 The command \line in the \CodeAfter ..................................... 23
   10.2 The command \SubMatrix in the \CodeAfter ............................... 24
11 The notes in the tabulars .......................................................... 25
   11.1 The footnotes ..................................................................... 25
   11.2 The notes of tabular ........................................................... 26
   11.3 Customisation of the tabular notes ........................................ 27
   11.4 Use of \{NiceTabular\} with threeparttable ............................... 29