The package **nicematrix**

F. Pantigny
fpantigny@wanadoo.fr
June 26, 2022

**Abstract**

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

\[
\begin{bmatrix}
\begin{array}{cccc}
C_1 & C_2 & \cdots & C_n \\
L_1 & a_{11} & a_{12} & \cdots & a_{1n} \\
L_2 & a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
L_n & a_{n1} & a_{n2} & \cdots & a_{nn}
\end{array}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>Product</th>
<th>dimensions (cm)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>L 3 l 5.5 h 1</td>
<td>30</td>
</tr>
<tr>
<td>standard</td>
<td>L 5.5 l 8 h 1.5</td>
<td>50.5</td>
</tr>
<tr>
<td>premium</td>
<td>L 8.5 l 10.5 h 2</td>
<td>80</td>
</tr>
<tr>
<td>extra</td>
<td>L 8.5 l 10 h 1.5</td>
<td>85.5</td>
</tr>
<tr>
<td>special</td>
<td>L 12 l 12 h 0.5</td>
<td>70</td>
</tr>
</tbody>
</table>

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

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

This package can be used with **xelatex**, **lualatex**, **pdflatex** but also by the classical workflow **latex-dvips-ps2pdf** (or Adobe Distiller). However, the file **nicematrix.dtx** of the present documentation should be compiled with **XeLaTeX**.

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

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

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

A command `\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).

---

\*This document corresponds to the version 6.10a of **nicematrix**, at the date of 2022/06/26.

\[1\] The latest version of the file **nicematrix.sty** may be downloaded from the `svn` server of TeXLive: https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty

\[2\] If you use Overleaf, Overleaf will do automatically the right number of compilations.
1  The environments of this package

The package nicematrix defines the following new environments.

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

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

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

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

The environment \{NiceTabularX\} is similar to the environment \{tabularx\} from the eponymous package.³

It’s recommended to use primarily the classical environments and to use the environments of nicematrix only when some feature provided by these environments is used (this will save memory).

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

2  The vertical space between the rows

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

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

Inspired by the package cellspace which deals with that problem, the package nicematrix provides two keys cell-space-top-limit and cell-space-bottom-limit similar to the parameters \cellspacetoplimit and \cellspacebottomlimit of cellspace.

There is also a key 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 nicematrix the same behaviour as those of array and amsmath. However, a value of 1 pt would probably be a good choice and we suggest to set them with \NiceMatrixOptions.⁴

\NiceMatrixOptions{cell-space-limits = 1pt}

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

³In fact, it’s possible to use directly the X columns in the environment \NiceTabular (and the required width for the tabular is fixed by the key width): cf. p. 21

⁴One should remark that these parameters apply also to the columns of type S of siunitx whereas the package cellspace is not able to act on such columns of type S.
## 3 The vertical position of the arrays

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

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

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

However, it’s also possible to use the tools of \texttt{booktabs}⁵: \texttt{\toprule}, \texttt{\bottomrule}, \texttt{\midrule}, etc.

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

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 \textit{following} the horizontal rule.

\begin{verbatim}
\NiceMatrixOptions{cell-space-limits=1pt}
\begin{enumerate}
\item an item
\item \begin{NiceArray}{cc|cc}[baseline=\text{line-3}]
\dfrac{1}{A} & \dfrac{1}{B} & 0 & 0 \\
\dfrac{1}{C} & \dfrac{1}{D} & 0 & 0 \\
\hline
0 & 0 & A & B \\
0 & 0 & D & D
\end{NiceArray}
\end{enumerate}
\end{verbatim}

³The extension \texttt{booktabs} is \textit{not} loaded by \texttt{nicematrix}. 

3
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.\footnote{The spaces after a command \Block are deleted.}

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 \textit{i-j} where \textit{i} is the number of rows of the block and \textit{j} its number of columns.
  
  If this argument is empty, its default value is \textit{1-1}. If the number of rows is not specified, or equal to \textit{*}, 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}, \{NiceTabular*\} and \{NiceTabularX\}, 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}{cw{c}{1cm}c|c}[margin]
\Block{3-3}<\Large>{A} & 0 \ldots & 0 \\
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}

\begin{pmatrix}
A & 0 \\
\iddots & \iddots & \iddots \\
0 & \ldots & 0 & 0
\end{pmatrix}

One may wish to raise the size of the “\textit{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 \textit{\large}, \textit{\Large} and \textit{\LARGE}. That’s why the command \textit{\Block} provides an option between angle brackets to specify some TEX code which will be inserted before the beginning of the math mode.\footnote{This argument between angular brackets may also be used to insert a command of font such as \textit{\bfseries} when the command \\
   is used in the content of the block.}

\begin{bNiceArray}{cw{c}{1cm}c|c}[margin]
\Block{3-3}<\Large>{A} & 0 \ldots & 0 \\
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}

\begin{pmatrix}
A & 0 \\
\iddots & \iddots & \iddots \\
0 & \ldots & 0 & 0
\end{pmatrix}

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

\begin{bNiceArray}{cw{c}{1cm}c|c}[margin]
\Block[r]{3-3}<\LARGE>{A} & 0 \ldots & 0 \\
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}

\begin{pmatrix}
A & 0 \\
\iddots & \iddots & \iddots \\
0 & \ldots & 0 & 0
\end{pmatrix}

In fact, the command \textit{\Block} accepts as first optional argument (between square brackets) a list of couples \textit{key=value}. The available keys are as follows:
• the keys \texttt{l}, \texttt{c} and \texttt{r} are used to fix the horizontal position of the content of the block, as explained previously;

• the key \texttt{fill} takes in as value a color and fills the block with that color;

• 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);

• 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;

• the keys \texttt{hlines}, \texttt{vlines} and \texttt{hvlines} draw all the corresponding rules in the block;

• the key \texttt{line-width} is the width of the rules (this key is meaningful only when one of the keys \texttt{draw}, \texttt{hvlines}, \texttt{vlines} and \texttt{hlines} is used);

• 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\footnote{This value is the initial value of the \texttt{rounded corners} of Tikz.});

• the keys \texttt{t} and \texttt{b} fix the base line that will be given to the block when it has a multi-line content (the lines are separated by \texttt{\backslash});

• when the key \texttt{tikz} is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz\footnote{Tikz should be loaded (by default, \texttt{ninematrix} only loads \texttt{pgf}) and, if it’s not, an error will be raised.} by using as options the value of that key \texttt{tikz} (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 47;

• the key \texttt{name} provides a name to the rectangular Tikz node corresponding to the block; it’s possible to use that name with Tikz in the \texttt{\CodeAfter} of the environment (cf. p. 28);

• the key \texttt{respect-arraystretch} prevents the setting of \texttt{\arraystretch} to 1 at the beginning of the block (which is the behaviour by default);

• 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}: it’s possible, in fact, in the list which is the value of the key \texttt{borders}, to add an entry of the form \texttt{tikz={list}} where \texttt{list} is a list of couples \texttt{key=value} of Tikz specifying the graphical characteristics of the lines that will be drawn (for an example, see p. 51).

One must remark that, by default, the commands \texttt{\Blocks} \textbf{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\{\ldots\}} of \texttt{array}).

\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose & tulip & daisy & dahlia \ \violet \\
& \Block[draw=red,fill=\[RGB\]{204,204,255},rounded-corners]\{2-2\} \\
&& \{\LARGE Some beautiful flowers\} \\
& & marigold \ \iris & & & lis \ \arum & & & forget-me-not & & & hyacinth \\
\end{NiceTabular}
4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.
- In the columns with a fixed width (columns $w\{\ldots\}$, $p\{\ldots\}$, $b\{\ldots\}$, $m\{\ldots\}$ and $X$), the content of the block is formatted as a paragraph of that width.
- The specification of the horizontal position provided by the type of column ($c$, $r$ or $l$) is taken into account for the blocks.
- The specifications of font specified for the column by a construction $>{\ldots}$ in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

\begin{NiceTabular}{@{}lr@{}} \hline
  \Block{2-1}{John} & 12 \\ & 13 \hline
  Steph & 8 \hline
  \Block{3-1}{Sarah} & 18 \\ & 17 \hline & 15 \hline
  Ashley & 20 \hline
  Henry & 14 \hline
  \Block{2-1}{Madison} & 15 \\ & 19 \hline
\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 `\` 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 to draw a frame around the cell with the key \texttt{draw} of the command \texttt{Block} and to fill the background with rounded corners with the keys \texttt{fill} and \texttt{rounded-corners}.\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{colorcell-like} is used).}
- It’s possible to draw one or several borders of the cell with the key \texttt{borders}.
We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.\footnote{One may consider that the default value of the first mandatory argument of `\Block` is 1-1.}

### 4.5 Horizontal position of the content of the block

By default, the horizontal position of the content of a block is computed by using the positions of the *contents* of the columns implied in that block. That’s why, in the following example, the header “First group” is correctly centered despite the instruction `!{\qquad}` in the preamble which has been used to increase the space between the columns (this is not the behaviour of `\multicolumn`).

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

In order to have an horizontal positioning of the content of the block computed with the limits of the columns of the LaTeX array (and not with the contents of those columns), one may use the key L, R and C of the command `\Block`.

### 5 The rules

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

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

\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 \Block: cf. p. 4) nor in the corners (created by the key \texttt{corner}: cf. p. 10).

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

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

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

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

5.1.2 The command \texttt{\cline}

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

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

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

\begin{NiceTabular}{cccc} \hline
A & B & C & D \\
\cline{2} \\
A & B & C & D \\
\end{NiceTabular}

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

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

In the environments of \texttt{nicematrix}, an instruction \texttt{\cline{i}} is equivalent to \texttt{\cline{i\text{-}i}}.
5.2 The thickness and the color of the rules

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

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

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

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

5.3 The tools of \texttt{nicematrix} for the rules

Here are the tools provided by \texttt{nicematrix} for the rules.

- the keys \texttt{hlines}, \texttt{vlines}, \texttt{hlines} and \texttt{hvlines-except-borders};
- the specifier “|” in the preamble (for the environments with preamble);
- the command \texttt{\Hline}.

All these tools don’t draw the rules in the blocks nor in the empty corners (when the key \texttt{corners} is used).

- These blocks are:
  - the blocks created by the command \texttt{\Block} \footnote{And also the command \texttt{\multicolumn} but it’s recommended to use instead \texttt{\Block} in the environments of \texttt{nicematrix}.} presented p. 4;
  - the blocks implicitly delimited by the continuous dotted lines created by \texttt{\Cdots}, \texttt{\Vdots}, etc. (cf. p. 23).
- The corners are created by the key \texttt{corners} explained below (see p. 10).

In particular, this remark explains the difference between the standard command \texttt{\hline} and the command \texttt{\Hline} provided by \texttt{nicematrix}.

5.3.1 The keys \texttt{hlines} and \texttt{vlines}

The keys \texttt{hlines} and \texttt{vlines} (which draw, of course, horizontal and vertical rules) take in as value a list of numbers which are the numbers of the rules to draw. \footnote{It’s possible to put in that list some intervals of integers with the syntax \texttt{i--j}.}

In fact, for the environments with delimiters (such as \texttt{\pNiceMatrix} or \texttt{\bNiceArray}), the key \texttt{vlines} don’t draw the exterior rules (this is certainly the expected behaviour).

\begin{verbatim}
$\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{pNiceMatrix}$
\end{verbatim}
5.3.2 The keys hvlines and hvlines-except-borders

The key hvlines (no value) is the conjunction of the keys hlines and vlines.

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

The key hvlines-except-borders is similar to the key hvlines but does not draw the rules on the horizontal and vertical borders of the array.

5.3.3 The (empty) corners

The four corners of an array will be designed by NW, SW, NE and SE (north west, south west, north east and south east).

For each of these corners, we will call empty corner (or simply corner) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.\footnote{For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural. The precise definition of a “non-empty cell” is given below (cf. p. 46).}

However, it’s possible, for a cell without content, to require nicematrix to consider that cell as not empty with the key \NotEmpty.

\NiceMatrixOptions{cell-space-top-limit=3pt}
\begin{NiceTabular}{*{6}{c}}[corners, hvlines]
& & & & A \\
& & & A & A & A \\
end{NiceTabular}
It's also possible to provide to the key `corners` a (comma-separated) list of corners (designed by NW, SW, NE and SE).

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

▷ The corners are also taken into account by the tools provided by nicematrix to color cells, rows and columns. These tools don't color the cells which are in the corners (cf. p. 14).

5.4 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. $^{15}$.

\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`.

5.5 Commands for customized rules  

It's also possible to define commands and letters for customized rules with the key `custom-line` available in `\NiceMatrixOptions` and in the options of individual environments. That key takes in as argument a list of `key=value` pairs. First, there is two keys to define the tools which will be used to use that new type of rule.

- the key `command` is the name (without the backslashes) of a command that will be created by nicematrix and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- the key `letter` takes in as argument a letter $^{16}$ that the user will use in the preamble of an environment with preamble (such as `{NiceTabular}`) in order to specify a vertical rule.

For the description of the rule itself, there is three possibilities.

- **First possibility**
  It's possible to specify composite rules, with a color and a color for the inter-rule space (as possible with `colortbl` for instance).
  - the key `multiplicity` is the number to consecutive rules that will be drawn: for instance, a value of 2 will create double rules such those created by `\hline\hline` or `||` in the preamble of an environment;
  - the key `color` sets the color of the rule ;

---

$^{15}$The author of this document considers that type of construction as graphically poor.

$^{16}$The following letters are forbidden: `!rmpbVX()[]@<>`
the key `sep-color` sets the color between two successive rules (should be used only in conjunction with `multiplicity`).

That system may be used, in particular, for the definition of commands and letters to draw rules with a specific color (and those rules will respect the blocks and corners as do all the rules of `nicematrix`).

```latex
\begin{NiceTabular}{lcIcIc}
\[custom-line = \{letter=I, color=blue\}\]
\hline
& \Block{1-3}{dimensions} \\ 
& L & l & h \\ 
\hline
Product A & 3 & 1 & 2 \\ 
Product B & 1 & 3 & 4 \\ 
Product C & 5 & 4 & 1 \\ 
\hline
\end{NiceTabular}
```

- **Second possibility**

It’s possible to use the key `tikz` (if Tikz is loaded). In that case, the rule is drawn directly with Tikz by using as parameters the value of the key `tikz` which must be a list of `key=value` pairs which may be applied to a Tikz path.

By default, no space is reserved for the rule that will be drawn with Tikz. It is possible to specify a reservation (horizontal for a vertical rule and vertical for an horizontal one) with the key `total-width`. That value of that key, is, in some ways, the width of the rule that will be drawn (`nicematrix` does not compute that width from the characteristics of the rule specified in `tikz`).

<table>
<thead>
<tr>
<th></th>
<th>dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
</tr>
<tr>
<td>Product A</td>
<td>3</td>
</tr>
<tr>
<td>Product B</td>
<td>1</td>
</tr>
<tr>
<td>Product C</td>
<td>5</td>
</tr>
</tbody>
</table>

Here is an example with the key `dotted` of Tikz.

```latex
\NiceMatrixOptions
\{ 
  custom-line = 
  \{ 
    letter = I , 
    tikz = dotted , 
    total-width = \pgflinewidth 
  \} 
\}
\begin{NiceTabular}{cIcIc}
  one & two & three \\ 
  four & five & six \\ 
  seven & eight & nine 
\end{NiceTabular}
```

- **Third possibility**: the key `dotted`

As one can see, the dots of a dotted line of Tikz have the shape of a square, and not a circle. That’s why the extension `nicematrix` provides in the key `custom-line` a key `dotted` which will
draw rounded dots. The initial value of the key `total-width` is, in this case, equal to the diameter of the dots (but the user may change the value with the key `total-width` if needed). Those dotted rules are also used by `nicematrix` to draw continuous dotted rules between cells of the matrix with \cddots, \ddots, etc. (cf. p. 23).

In fact, `nicematrix` defines by default the command \hdottedline and the letter “:” for those dotted rules.\footnote{However, it's possible to overwrite those definitions with a `custom-line` (in order, for example, to switch to dashed lines).}

\begin{NiceMatrixOptions}
\customline{letter = :, command = hdottedline, dotted}
\end{NiceMatrixOptions}

Thus, it's possible to use the command \hdottedline to draw a horizontal dotted rule.

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

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

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

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

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

6 The color of the rows and columns

6.1 Use of colortbl

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

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

- The package colortbl patches array, leading to some incompatibilities (for instance with the command \hdotsfor).

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

As for this phenomenon, Adobe Reader gives better results than MuPDF.

The package \texttt{nicematrix} provides tools to avoid those problems.

6.2 The tools of \texttt{nicematrix} in the `\CodeBefore`

The package \texttt{nicematrix} provides some tools (independent of \texttt{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.\footnote{If you use Overleaf, Overleaf will do automatically the right number of compilations.}

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

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

\begin{verbatim}
\begin{pNiceArray}{preamble}
\CodeBefore
  instructions of the code-before
\Body
  contents of the environment
\end{pNiceArray}
\end{verbatim}

New commands are available in that \texttt{\CodeBefore}: \texttt{\cellcolor}, \texttt{\rectanglecolor}, \texttt{\rowcolor}, \texttt{\columncolor}, \texttt{\rowcolors}, \texttt{\rowlistcolors}, \texttt{\chessboardcolors} and \texttt{\arraycolor}.\footnote{Remark that, in the \texttt{\CodeBefore}, PGF/Tikz nodes of the form “(i-|j)” are also available to indicate the position to the potential rules: cf. p. 43.}

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.

These commands don’t color the cells which are in the “corners” if the key \texttt{corners} is used. This key has been described p. 10.

- The command \texttt{\cellcolor} takes its name from the command \texttt{\cellcolor} of \texttt{colortbl}.

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

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

- The command \texttt{\rectanglecolor} takes three mandatory arguments. The first is the color. The second is the upper-left cell of the rectangle and the third is the lower-right cell of the rectangle.
\begin{NiceTabular}{|c|c|c|}
\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 \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. 21). It’s only a particular case of \texttt{\rectanglecolor}.

- 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}

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

We have used the key \texttt{r} which aligns all the columns rightwards (cf. p. 37).

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

\begin{NiceArray}{lll}[hvlines]
\CodeBefore
\rowcolor{red!15}{1,3-5,8-}
\Body
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}$

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

- The command \texttt{\rowcolors} (with a \texttt{s}) takes its name from the command \texttt{\rowcolors} of \texttt{xcolor}.\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}.} The \texttt{s} emphasizes the fact that there is \texttt{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 \rowcolors (an interval of the form \textit{i-j} describes in fact the interval of all the rows of the tabular, beginning with the row \textit{i}).

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

- The key \textit{cols} describes a set of columns. The command \rowcolors will color only the cells of these columns. The value is a comma-separated list of intervals of the form \textit{i-j} (where \textit{i} or \textit{j} may be replaced by *).
- With the key \textit{restart}, each interval of rows (specified by the first mandatory argument) begins with the same color.\footnote{Otherwise, the color of a given row relies only upon the parity of its absolute number.}
- With the key \textit{respect-blocks} the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command \Block: cf. p. 4).

\begin{NiceTabular}{clr}[hvlines]\
\CodeBefore
\rowcolors[gray]{2}{0.8}{}[cols=2-3,restart]\
\Body
\Block{1-*}{Results} \& John & 12 \\\n& Stephen & 8 \\
& Sarah & 18 \\
& Ashley & 20 \\
& Henry & 14 \\
& Madison & 15 \\
\end{NiceTabular}

\begin{NiceTabular}{lr}[hvlines]\
\CodeBefore
\rowcolors{1}{blue!10}{}[respect-blocks]\
\Body
\Block{2-1}{John} & 12 \\
& 13 \\
& 13 \\
& 8 \\
\Block{3-1}{Sarah} & 18 \\
& 17 \\
& 15 \\
& 20 \\
& 18 \\
& 17 \\
& 15 \\
& 14 \\
\Block{2-1}{Madison} & 15 \\
& 19 \\
\end{NiceTabular}

- The extension \nicematrix provides also a command \rowlistcolors. This command generalises the command \rowcolors: instead of two successive arguments for the colors, this command takes in an argument which is a (comma-separated) list of colors. In that list, the symbol = represent a color identical to the previous one.
We recall that all the color commands we have described don’t color the cells which are in the “corners”. In the following example, we use the key \texttt{corners} to require the determination of the corner \textit{north east} (NE).

\begin{NiceTabular}{cccccc} \[ \texttt{corners=NE,margin,hvlines,first-row,first-col} \] \CodeBefore \rowlistcolors{1}{blue!15,} \Body & 0 & 1 & 2 & 3 & 4 & 5 & 6 \0 & 1 \1 & 1 & 1 \2 & 1 & 2 & 1 \3 & 1 & 3 & 3 & 1 \4 & 1 & 4 & 6 & 4 & 1 \5 & 1 & 5 & 10 & 10 & 5 & 1 \6 & 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{NiceTabular}

One should remark that all the previous 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}.

\begin{NiceTabular}{c}{lSSSS} \CodeBefore \rowcolor{red!15}{1-2} \rowcolors{3}{blue!15}{ } \Body \toprule \Block{2-1}{Product} & \Block{1-3}{dimensions (cm)} & & & \Block{2-1}{otate Price} \cmidrule(rl){2-4} & L & l & h \midrule small & 3 & 5.5 & 1 & 30 \standard & 5.5 & 8 & 1.5 & 50.5 \premium & 8.5 & 10.5 & 2 & 80 \extra & 8.5 & 10 & 1.5 & 85.5 \special & 12 & 12 & 0.5 & 70 \bottomrule \end{NiceTabular}

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

\begin{NiceTabular}{c} \CodeBefore \rowlistcolors{1}{red!15,blue!15,green!15} \Body Peter \\ James \\ Abigail \\ Elisabeth \\ Claudius \\ Jane \\ Alexandra \\
\end{NiceTabular}
6.3 Color tools with the syntax of colortbl

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

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

- \texttt{\cellcolor} which colorizes a cell;\footnote{However, this command \texttt{\cellcolor} will delete the following spaces, which does not the command \texttt{\cellcolor} of colortbl.}
- \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 colortbl (however, unlike the command \texttt{\columncolor} of colortbl, this command \texttt{\columncolor} can appear within another command, itself used in the preamble of the array).

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

7 The command \texttt{\RowStyle}

The command \texttt{\RowStyle} takes in as argument some formatting instructions that will be applied to each cell on the rest of the current row.

That command also takes in as optional argument (between square brackets) a list of key=value pairs.

- The key \texttt{nb-rows} sets the number of rows to which the specifications of the current command will apply (with the special value *, it will apply to all the following rows).
- The keys \texttt{cell-space-top-limit}, \texttt{cell-space-bottom-limit} and \texttt{cell-space-limits} are available with the same meaning that the corresponding global keys (cf. p. 2).
- The key \texttt{rowcolor} sets the color of the background and the key \texttt{color} sets the color of the text.\footnote{The key \texttt{color} uses the command \texttt{\color} but inserts also an instruction \texttt{\leavevmode} before. This instruction prevents a extra vertical space in the cells which belong to columns of type \texttt{p}, \texttt{b}, \texttt{m} and \texttt{X} (which start in vertical mode).}
• The key \textbf{bold} enforces bold characters for the cells of the row, both in math and text mode.

\begin{NiceTabular}{cccc}
\hline
\RowStyle[cell-space-limits=3pt]{\rotate}
first & second & third & fourth \\
\RowStyle[\nb-rows=2,\rowcolor=blue!50,\color=white]{\sffamily}
1 & 2 & 3 & 4 \\
I & II & III & IV \\
\end{NiceTabular}

The command \texttt{rotate} is described p. 37.

8 The width of the columns

8.1 Basic tools

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

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

In the environments of \texttt{nicematrix}, it’s also possible to fix the \textit{minimal} width of all the columns (excepted the potential exterior columns: cf. p. 21) directly with the key \texttt{columns-width}.

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

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

\begin{pNiceMatrix}[\texttt{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 arrays of a current scope with the command \texttt{NiceMatrixOptions}.

\NiceMatrixOptions{\texttt{columns-width=10mm}}

\begin{pNiceMatrix}
a & b \\
c & d \\
\end{pNiceMatrix} = \begin{pNiceMatrix}
a & 1245 \\
c & 2 \\
\end{pNiceMatrix}
But it’s also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment \texttt{\{NiceMatrixBlock\}} with the option \texttt{auto-columns-width}. 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}
\begin{bNiceMatrix}
  1 & 1245345 \\ 345 & 2 \\
\end{bNiceMatrix}
\end{NiceMatrixBlock}

8.2 The columns \texttt{V} of varwidth

Let’s recall first the behaviour of the environment \texttt{\{varwidth\}} of the eponymous package varwidth. That environment is similar to the classical environment \texttt{\{minipage\}} but the width provided in the argument is only the maximal width of the created box. In the general case, the width of the box constructed by an environment \texttt{\{varwidth\}} is the natural width of its contents.

That point is illustrated on the following examples.

\begin{NiceTabular}[corners=NW,hvlines]{V{3cm}V{3cm}V{3cm}}
  & some very very very long text & some very very very long text \\
  some very very very long text & \\
  some very very very long text & \\
\end{NiceTabular}

The package varwidth provides also the column type \texttt{V}. A column of type \texttt{V\{\langle dim\rangle\}} encapsulates all its cells in a \texttt{\{varwidth\}} with the argument \texttt{\langle dim\rangle} (and does also some tuning).

When the package varwidth is loaded, the columns \texttt{V} of varwidth are supported by nicematrix. Concerning nicematrix, one of the interests of this type of columns is that, for a cell of a column of type \texttt{V}, the PGF/Tikz node created by nicematrix for the content of that cell has a width adjusted to the content of the cell : cf. p. 41. If the content of the cell is empty, the cell will be considered as empty by nicematrix in the construction of the dotted lines and the «empty corners» (that’s not the case with a cell with a column \texttt{p}, \texttt{m} or \texttt{b}).

At this time, this is the only usage of the environment \texttt{\{NiceMatrixBlock\}} but it may have other usages in the future.
Some very very very long text

\begin{NiceTabular}[width=9cm]{X[2]X}[hvlines]
\hline
\text{a rather long text which fits on several lines} & \text{a rather long text which fits on several lines} \\
\hline
\text{a shorter text} & \text{a shorter text}
\end{NiceTabular}

8.3 The columns X

The environment \{NiceTabular\} provides X columns similar to those provided by the environment \{tabularx\} of the eponymous package.

The required width of the tabular may be specified with the key width (in \{NiceTabular\} or in \NiceMatrixOptions). The initial value of this parameter is \linewidth (and not \textwidth).

For sake of similarity with the environment \{tabularx\}, nicematrix also provides an environment \{NiceTabularX\} with a first mandatory argument which is the width of the tabular.\footnote{If tabularx is loaded, one must use \{NiceTabularX\} (and not \{NiceTabular\}) in order to use the columns X (this point comes from a conflict in the definitions of the specifier X).}

As with the packages tabu and tabulararray, the specifier X takes in an optional argument (between square brackets) which is a list of keys.

- It’s possible to give a weight for the column by providing a positive integer directly as argument of the specifier X. For example, a column X[2] will have a width double of the width of a column X (which has a weight equal to 1).\footnote{The negative values of the weight, as provided by tabu (which is now obsolete), are not supported by nicematrix. If such a value is used, an error will be raised.}
- It’s possible to specify an horizontal alignment with one of the letters l, c and r (which insert respectively \raggedright, \centering and \raggedleft followed by \arraybackslash).
- It’s possible to specify a vertical alignment with one of the keys t (alias p), m and b (which construct respectively columns of type p, m and b). The default value is t.

9 The exterior rows and columns

The options first-row, last-row, first-col and last-col allow the composition of exterior rows and columns in the environments of nicematrix. It’s particularly interesting for the (mathematical) matrices.

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

---

21
The dotted lines have been drawn with the tools presented p. 23.

We have several remarks to do.

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

- One may wonder how nicematrix determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
  
  - For the environments with explicit preamble, like \{NiceTabular\} and \{pNiceArray\}, the number of columns can obviously be computed from the preamble.
  
  - When the option \texttt{light-syntax} (cf. p. 39) is used, nicematrix has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option \texttt{light-syntax}). The analysis of this whole body gives the number of rows (but not the number of columns).
  
  - In the other cases, nicematrix compute the number of rows and columns during the first compilation and write the result in the \texttt{aux} file for the next run. However, it’s possible to provide the number of the last row and the number of the last column as values of the options \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}, \texttt{code-for-first-col} and \texttt{code-for-last-col}. These options specify tokens that will be inserted before each cell of the corresponding row or column.

```latex
\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}}
```

29The users wishing exterior columns with another type of alignment should consider the command \SubMatrix available in the \texttt{CodeAfter} (cf. p. 29).
\begin{bNiceMatrix}
a_{31} & a_{32} & a_{33} & a_{34} & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
& C_1 & \Cdots & & C_4 &
\end{bNiceMatrix}$

```
\begin{bNiceMatrix}
C_1 & \dots & \dots & C_4 \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
& a_{21} & a_{22} & a_{23} & a_{24} & \vdots \\
L_4 & a_{31} & a_{32} & a_{33} & a_{34} & \vdots \\
& a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
C_1 & \dots & \dots & C_4
\end{bNiceMatrix}
```

Remarks

- As shown in the previous example, the horizontal and vertical rules don’t extend in the exterior rows and columns. This remark also applies to the customized rules created by the key `custom-line` (cf. p. 11).
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that 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. 19) 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. 29.

10 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`.

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

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

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

Footnotes:
- The command `\Iddots`, defined in `nicematrix`, is a variant of `\ddots` with dots going forward. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of `unicode-math`.
- The precise definition of a “non-empty cell” is given below (cf. p. 46).
- 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. 27.
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 \texttt{nicematrix}:

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

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

In the first column of this example, there are two instructions \texttt{\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 & & 0 \\
  \Vdots & & & \\
  & & & \Vdots \\
  0 & & \Cdots & 0
\end{bNiceMatrix}

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

There are also other means to change the size of the matrix. Someone might want to use the optional
argument of the command \texttt{\Vdots} for the vertical dimension and a command \texttt{\hspace{*}} in a cell for the
horizontal dimension.\footnote{In \texttt{nicematrix}, one should use \texttt{\hspace{*}} and not \texttt{\hspace} for such an usage because \texttt{nicematrix} loads \texttt{array}. One may also remark that it’s possible to fix the width of a column by using the environment \texttt{\NiceArray} (or one of its variants) with a column of type \texttt{w} or \texttt{W}; see p. 19}

However, a command \texttt{\hspace{*}} might interfere with the construction of the dotted lines. That’s why
the package \texttt{nicematrix} provides a command \texttt{\Hspace} which is a variant of \texttt{\hspace} transparent for
the dotted lines of \texttt{nicematrix}.

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

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

10.1 The option \texttt{nullify-dots}

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

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

If we add \texttt{\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 \texttt{nicematrix}, if we replace \texttt{\pmatrix} by \texttt{\pNiceMatrix} and \texttt{\ldots} by \texttt{\Ldots}, the
geometry of the matrix is not changed.

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

$A = \begin{pmatrix} h & i & j & k & l & m \\ x & \ldots & \ldots & \ldots & \ldots & x \end{pmatrix}$
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 \texttt{nullify-dots} (and only one instruction \texttt{\Ldots} is necessary).

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

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

### 10.2 The commands \texttt{\Hdotsfor} and \texttt{\Vdotsfor}

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

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

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

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

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

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

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

\[
\begin{bNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n] \\
\Vdots & \Ddots & \Vdots \\
C[a_n,a_1] & \Cdots & C[a_n,a_n] \\
\Vdots & \Ddots & \Vdots \\
\end{bNiceMatrix}
\]

We recall that when \texttt{xcolor} is loaded with the option \texttt{table}, the package \texttt{colortbl} is loaded.

\begin{verbatim}
\begin{bNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n] \\
\& \hspace*{20mm} & \Cdots & C[a_1,a_n] \\
\end{bNiceMatrix}
\end{verbatim}
10.3 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 \texttt{nicematrix} without having to modify the code of each matrix. It’s possible with the keys \texttt{renew-dots} and \texttt{renew-matrix}.\footnote{The options \texttt{renew-dots}, \texttt{renew-matrix} can be fixed with the command \texttt{\NiceMatrixOptions} like the other options. However, they can also be fixed as options of the command \texttt{\usepackage}.}

- The option \texttt{renew-dots}
  
  With this option, the commands \texttt{\ldots}, \texttt{\cdots}, \texttt{\vdots}, \texttt{\ddots}, \texttt{\iddots} and \texttt{\hdotsfor} are redefined within the environments provided by \texttt{nicematrix} and behave like \texttt{\Ldots}, \texttt{\Cdots}, \texttt{\Vdots}, \texttt{\Ddots}, \texttt{\Iddots} and \texttt{\Hdots}; the command \texttt{\ldots} (“automatic dots” of \texttt{amsmath}) is also redefined to behave like \texttt{\Ldots}.

- The option \texttt{renew-matrix}
  
  With this option, the environment \texttt{\{matrix\}} is redefined and behave like \texttt{\{NiceMatrix\}}, and so on for the five variants.

Therefore, with the keys \texttt{renew-dots} and \texttt{renew-matrix}, a classical code gives directly the output of \texttt{nicematrix}.

\begin{verbatim}
\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{verbatim}

10.4 The labels of the dotted lines

The commands \texttt{\ldots}, \texttt{\cdots}, \texttt{\vdots}, \texttt{\ddots}, \texttt{\iddots} and \texttt{\hdotsfor} (and the command \texttt{\line} in the \texttt{\CodeAfter} which is described p. 29) 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 \texttt{scriptstyle}.

\begin{verbatim}
\begin{bNiceMatrix}
1 & \hspace*{1cm} & 0 \\
\Ddots^{n \text{ times}} & & \\
0 & & 1
\end{bNiceMatrix}
\end{verbatim}
10.5 Customisation of the dotted lines

The dotted lines drawn by \Ldots, \Cdots, \Vdots, \Ddots, \Iddots, \Hdots for \and by the command \line in the \CodeAfter which is described p. 29 may be customized by the following options (specified between square brackets after the command):

- color;
- radius;
- shorten-start, shorten-end and shorten;
- inter;
- 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 (xdots to remind that it works for \Cdots, \Ldots, \Vdots, etc.), and, thus have for names:

- xdots/color;
- xdots/radius;
- xdots/shorten-start, xdots/shorten-end and xdots/shorten;
- xdots/inter;
- 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. 21.

New 6.9 The option xdots/radius

The option radius fixes the radius of the dots. The initial value is 0.53 pt.

The option xdots/shorten

The keys xdots/shorten-start and xdots/shorten-end fix the margin at the extremities of the line. The key xdots/shorten fixes both parameters. The initial value is 0.3 em (it is recommended to use a unit of length dependent of the current font).

New 6.10 The keys xdots/shorten-start and xdots/shorten-end have been introduced in version 6.10. In the previous versions, there was only xdots/shorten.

New 6.9 The option xdots/inter

The option xdots/inter fixes the length between the dots. The initial value is 0.45 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.}

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

In order to provide lines with rounded dots in the style of those provided by \dots (at least with the Computer Modern fonts), the package nicematrix embeds its own system to draw a dotted line
This style is called standard and that’s the initial value of the parameter xdots/line-style. However (when Tikz is loaded) it’s possible to use for xdots/line-style any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tikz paths (with the exception of “color”, “shorten >” and “shorten <”).

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

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

10.6 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, by the command \Hline, by the keys hlines, vlines, hvlines and hvlines-except-borders and by the tools created by custom-line are not drawn within the blocks).\(^{37}\)

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

11 The \textbackslash CodeAfter

The option code-after may be used to give some code that will be executed after the construction of the matrix.\(^{38}\)

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

The experienced users may, for instance, use the PGF/Tikz nodes created by nicematrix in the \CodeAfter. These nodes are described further beginning on p. 40.

Moreover, several special commands are available in the \CodeAfter: line, \SubMatrix, \OverBrace and \UnderBrace. We will now present these commands.

\(^{37}\)On the other side, the command \textbackslash Line in the \CodeAfter (cf. p. 29) does not create block.

\(^{38}\)There is also a key code-before described p. 14.
11.1 The command \line in the \CodeAfter

The command \line draws directly dotted lines between cells or blocks. It takes in two arguments for the cells or blocks to link. Both argument may be:

- a specification of cell of the form i-j where is the number of the row and j is the number of the column;
- \textbf{New 6.10} the name of a block (created by the command \Block with the key name of that command).

The options available for the customisation of the dotted lines created by \Cdots, \Vdots, etc. are also available for this command (cf. p. 27).

This command may be used, for example, to draw a dotted line between two adjacent cells.

\begin{verbatim}
\NiceMatrixOptions{xdots/shorten = 0.6 em}
\begin{pNiceMatrix}
I & 0 & \Cdots &0 \\
0 & I & \Ddots & \Vdots \\
\Vdots & \Ddots & I & 0 \\
0 & \Cdots & 0 & I
\end{pNiceMatrix}
\end{verbatim}

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by \Ddots are “parallelized”: cf. p. 45).

\begin{verbatim}
\begin{bNiceMatrix}
1 & \Cdots & & 1 & 2 & \Cdots & 2 \\
0 & \Ddots & & \Vdots & \Vdots & \hspace*{2.5cm} & \Vdots \\
\Vdots & \Ddots & & & & & \\
0 & \Cdots & 0 & 1 & 2 & \Cdots & 2
\end{bNiceMatrix}
\end{verbatim}

11.2 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.\footnote{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 (eg.: \SubMatrix[(2-2){4-7}]).}
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.

\[
\begin{NiceArray}{ccc@{\hspace{1.5em}}}c
1 & 1 & 1 \backslash & x \backslash
\dfrac{1}{4} & \dfrac{1}{2} & \dfrac{1}{4} \backslash & y \backslash
1 & 2 & 3 \end{NiceArray}
\]

In fact, the command \SubMatrix also takes in two optional arguments specified by the traditional symbols ^ and _ for material in superscript and subscript.

\[
\begin{bNiceMatrix}
1 & 1 & 1 \\
1 & a & b \\
1 & c & d \\
\end{bNiceMatrix}^{T}
\]

The options of the command \SubMatrix are as follows:

- **left-xshift** and **right-xshift** 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 (\ht + \depth);
- **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);
- **hlines** is similar to **vlines** but for the horizontal rules;
- **hvlines**, 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 \NiceMatrixOptions, at the level of the environments of nicematrix or as option of the command \CodeAfter with the prefix sub-matrix which means that their names are therefore sub-matrix/left-xshift, sub-matrix/right-xshift, sub-matrix/xshift, etc.
Here is the same example with the key \texttt{slim} used for one of the submatrices.

\[
\begin{NiceArray}{cc@{\hspace{5mm}}l}
& & \frac{1}{2} \\
& & \frac{1}{4} \\
a & b & \frac{1}{2}a+\frac{1}{4}b \\
c & d & \frac{1}{2}c+\frac{1}{4}d \\
\end{NiceArray}
\]

\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 \SubMatrix. That name is used to create PGF/Tikz nodes: cf p. 44.

It’s also possible to specify some delimiters\footnote{Those delimiters are \(\langle\), \([\), \{ and the closing ones. Of course, it’s also possible to put \(|\) and \(||\) in the preamble of the environment.} by placing them in the preamble of the environment (for the environments with a preamble: \{NiceArray\}, \{pNiceArray\}, etc.). This syntax is inspired by the extension blkarray.

When there are two successive delimiters (necessarily a closing one following by an opening one for another submatrix), a space equal to \texttt{\enskip} is automatically inserted.

\[
\begin{pNiceArray}{(c)(c)(c)}
a_{11} & a_{12} & a_{13} \\
a_{21} & \displaystyle \int_0^1\dfrac{1}{x^2+1}\,dx & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{pNiceArray}
\]

\[
\begin{pmatrix}
a_{11} \\ a_{21} \\ a_{31}
\end{pmatrix}
\begin{pmatrix}
a_{12} \\ 1 \\ 0
\end{pmatrix}
\begin{pmatrix}
\int_0^1 \\ x^2+1 \\ dx
\end{pmatrix}
\begin{pmatrix}
a_{13} \\ a_{23} \\ a_{33}
\end{pmatrix}
\]

11.3 The commands \texttt{\OverBrace} and \texttt{\UnderBrace} in the \texttt{\CodeAfter}

The commands \texttt{\OverBrace} and \texttt{\UnderBrace} provide a way to put horizontal braces on a part of the array. These commands take in three arguments:

- the first 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 second argument is the lower-right corner with the same syntax;
- the third argument is the label of the brace that will be put by \nicematrix\ (with PGF) above the brace (for the command \texttt{\OverBrace}) or under the brace (for \texttt{\UnderBrace}).

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16 \\
\end{pNiceMatrix}
\CodeAfter
\OverBrace{1-1}{2-3}{A}
\UnderBrace{1-4}{2-6}{B}
\end{pNiceMatrix}

In fact, the commands \texttt{\OverBrace} and \texttt{\UnderBrace} take in an optional argument (in first position and between square brackets) for a list of \texttt{key=value} pairs. The available keys are:
• **left-shorten** and **right-shorten** which do not take in value; when the key **left-shorten** is used, the abscissa of the left extremity of the brace is computed with the contents of the cells of the involved sub-array, otherwise, the position of the potential vertical rule is used (idem for **right-shorten**).

• **shorten**, which is the conjunction of the keys **left-shorten** and **right-shorten**;

• **yshift**, which shifts vertically the brace (and its label);

• **color**, which sets the color of the brace (and its label).

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16 \\
\CodeAfter
\OverBrace[shorten,yshift=3pt]{1-1}{2-3}{A}
\OverBrace[shorten,yshift=3pt]{1-4}{2-6}{B}
\end{pNiceMatrix}

\begin{NiceTabular}{@{}llr@{}}
\toprule 
\RowStyle{\bfseries}
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 \\
\addlinespace[1.5ex]
\end{NiceTabular}

12 The notes in the tabulars

12.1 The footnotes

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

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

If **nicematrix** is loaded with the option **footnotehyper**, the package **footnotehyper** is loaded (if it is not yet loaded) and it is used to extract footnotes.

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

12.2 The notes of tabular

The package **nicematrix** also provides a command \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 specified by **first-col** and **last-col**). With no surprise, that command is available only in the environments without delimiters, that is to say \{NiceTabular\}, \{NiceArray\} and \{NiceMatrix\}.

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

\begin{NiceTabular}{@{}llr@{}}
\toprule 
Last name & First name & Birth day \\
\midrule 
Achard\tabularnote{Achard is an old family of the Poitou.} & Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.} & Mathilde & 23 mai 1988 \\
\addlinespace[1.5ex]
\end{NiceTabular}
Achard is an old family of the Poitou.

The name Lefebvre is an alteration of the name Lefebure.

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

- If a command \texttt{\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 \texttt{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.

\textbf{New 6.8} If several commands \texttt{\tabularnote} are used in a tabular with the same argument, only one note is inserted at the end of the tabular (but all the labels are composed, of course). It’s possible to control that feature with the key \texttt{notes/detect-duplicates}.

- 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. 34. This table has been composed with the following code.

\begin{verbatim}
\begin{table}
\setlength{\belowcaptionskip}{1ex}
\centering
\caption{Use of \texttt{\tabularnote{It\textquotesingle s possible to put a note in the caption.}}}
\label{t:tabularnote}
\begin{NiceTabular}{@{}llc@{}}
\[\texttt{notes/bottomrule, tabularnote = Some text before the notes.}\]
\toprule
Last name & First name & Length of life \\
\midrule
Churchill & Wiston & 91 \\
Nightingale\tabularnote{Considered as the first nurse of history} & Florence\tabularnote{Nicknamed \textquoteleft\textquoteright the Lady with the Lamp\textquoteright.} & 90 \\
Schoelcher & Victor & 89\tabularnote{The label of the note is overlapping.}
\bottomrule
\end{NiceTabular}
\end{table}
\end{verbatim}
<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$^{b,c}$</td>
<td>Florence$^d$</td>
<td>90</td>
</tr>
<tr>
<td>Schoelcher</td>
<td>Victor</td>
<td>89$^c$</td>
</tr>
<tr>
<td>Touchet</td>
<td>Marie$^d$</td>
<td>89</td>
</tr>
<tr>
<td>Wallis</td>
<td>John</td>
<td>87</td>
</tr>
</tbody>
</table>

Some text before the notes.

$^a$ It’s possible to put a note in the caption.

$^b$ Considered as the first nurse of history.

$^c$ Nicknamed “the Lady with the Lamp”.

$^d$ This note is shared by two references.

$^e$ The label of the note is overlapping.

12.3 Customisation of the tabular notes

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

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

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

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

34
We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.
  Initial value: `false`
  That key is also available within a given environment.

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

- The key `notes/style` is a command whose argument is specified by `#1` and which gives the style of numerotation of the notes. That style will be used by `\ref` when referencing a tabular note marked with a command `\label`. The labels formatted by that style are used, separated by commas, when the user puts several consecutive commands `\tabularnote`. The marker `#1` is meant to be the name of a LaTeX counter.
  Initial value: `\textit{\alph{#1}}`
  Another possible value should be a mere `\arabic{#1}`

- The key `notes/label-in-tabular` is a command whose argument is specified by `#1` which is used when formatting the label of a note in the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.
  Initial value: `\textsuperscript{#1}`
  In French, it’s a tradition of putting a small space before the label of note. That tuning could be acheived by the following code:

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

- The key `notes/label-in-list` is a command whose argument is specified by `#1` which is used when formatting the label in the list of notes at the end of the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.
  Initial value: `\textsuperscript{#1}`
  In French, the labels of notes are not composed in upper position when composing the notes. Such behaviour could be acheived by:

  ```latex
  \NiceMatrixOptions{notes/label-in-list = #1.\nobreak\hspace{0.25em}}
  ```
  The command `\nobreak` is for the event that the option `para` is used.

- The notes are composed at the end of the tabular by using internally a style of list of `enumitem`. This style of list is defined as follows (with, of course, keys of `enumitem`):
  `noitemsep, leftmargin = *, align = left, labelsep = 0pt`
  The specification `align = left` in that style requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 34).
  The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that style of list (it uses internally the command `\setlist*` of `enumitem`).

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

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

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

• The key \texttt{notes/detect-duplicates} activates the detection of the commands \texttt{\tabularnotes} with the same argument.

  Initial value: \texttt{true}

For an example of customisation of the tabular notes, see p. \texttt{48}.

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

If you wish to use the environment \{NiceTabular\}, \{NiceTabular\} \{NiceTabularX\} 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*}\TPT@hookin{NiceTabularX}}
\makeatother
\end{verbatim}

13 Other features

14 Autres fonctionnalités

14.1 Command \texttt{\ShowCellNames}

\textbf{New 6.9} The command \texttt{\ShowCellNames}, which may be used in the \texttt{\CodeBefore} and in the \texttt{\CodeAfter} display the name (with the form \texttt{i-j}) of each cell.

\begin{verbatim}
\begin{NiceTabular}{ccc}[hvlines,cell-space-limits=3pt]
\CodeBefore \ShowCellNames
\CodeAfter
\Body
\Block{2-2}{ & & test \}
 & & blabla \ & some text & nothing
\end{NiceTabular}
\end{verbatim}

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

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

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

### 14.4 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}
$A = \begin{pNiceMatrix}[last-row=4, last-col=4]
e_1 & e_2 & e_3 \\
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3
\text{image of } e_1 & e_2 & e_3
\end{pNiceMatrix}$
\end{verbatim}

### 14.5 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}).

\footnote{It can also be used in \texttt{\RowStyle} (cf. p. 18.)}

37
\begin{bNiceArray}{cccc|c}
small, 
last-col, 
code-for-last-col = \scriptscriptstyle, 
columns-width = 3mm 
\end{bNiceArray}

$\begin{bNiceArray}[first-row,
first-col, 
code-for-first-row = \mathbf{\alpha{jCol}} , 
code-for-first-col = \mathbf{\arabic{iRow}} ]
\end{bNiceArray}$

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

### 14.6 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{\CodeBefore} (cf. p. 14) and in the \texttt{\CodeAfter} (cf. p. 28), \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}
\end{pNiceMatrix}
\end{verbatim}
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.

```latex
\begin{NiceMatrix}
[light-syntax,first-row,first-col]
{} & a & b \\
{a} & 2\cos a & \{\cos a + \cos b\} \\
b & \cos a + \cos b & \{2 \cos b\}
\end{NiceMatrix}
```

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

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

### 14.8 Color of the delimiters

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

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

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 29).

### 14.9 The environment {NiceArrayWithDelims}

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

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

---

43The 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.
14.10 The command \OnlyMainNiceMatrix

The command \OnlyMainNiceMatrix executes its argument only when it is in the main part of the array, that is to say it is not in one of the exterior rows. If it is used outside an environment of \nicematrix, that command is no-op.

For an example of utilisation, see tex.stackexchange.com/questions/488566

15 Use of Tikz with \nicematrix

15.1 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).

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. 44

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

In the environment with the number n, the node of the row i and column j has for name nm-n-i-j.

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

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

In the \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. 55).

---

44 One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 23) and the computation of the “corners” (cf. p. 10).
The nodes of the last column (excepted the potential «last column» specified by \texttt{last-col}) may also be indicated by \texttt{i-last}. Similarly, the nodes of the last row may be indicated by \texttt{last-j}.

15.1.1 The columns \textbf{V} of \texttt{varwidth}

When the extension \texttt{varwidth} is loaded, the columns of the type \texttt{V} defined by \texttt{varwidth} are supported by \texttt{nicematrix}. It may be interessant to notice that, for a cell of a column of type \texttt{V}, the PGF/Tikz node created by \texttt{nicematrix} for the content of that cell has a width adjusted to the content of the cell. This is in contrast to the case of the columns of type \texttt{p}, \texttt{m} or \texttt{b} for which the nodes have always a width equal to the width of the column. In the following example, the command \texttt{\lipsum} is provided by the eponymous package.

\begin{NiceTabular}{V{10cm}}
  \bfseries \large \\
  \lipsum[1][1-4] \\
  \CodeAfter \\
  \tikz \draw [rounded corners] (1-1) -| (last-|2) -- (last-|1) |- (1-1) ; \\
\end{NiceTabular}

\begin{tabular}{|c|c|c|}
  \hline
  \textbf{Titre} \\
  \hline
  Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. \\
  \hline
\end{tabular}

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 43).

15.2 The “medium nodes” and the “large nodes”

In fact, the package \texttt{nicematrix} can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option \texttt{create-medium-nodes} and the second ones with the option \texttt{create-large-nodes}.$^45$

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

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

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.$^46$

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

$^{45}$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}.

$^{46}$There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 21).
The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.

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

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

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

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

Here is an array composed with the following code:

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

Here, we have colored all the cells of the array with `\chessboardcolors`.

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

The nodes we have described are not available by default in the `\CodeBefore` (described p. 14). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the array by using informations written on the aux file and created a second time during the construction of the array itself).

Here is an example which uses these nodes in the `\CodeAfter`.

\[
\begin{NiceArray}{c@{\;}c@{\;}c@{\;}c@{\;}c}[create-medium-nodes]
  u_1 &-& u_0 &=& r \\
  u_2 &-& u_1 &=& r
\end{NiceArray}
\]

\[47\text{The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.}\]
\begin{align*}
\phantom{u_5} & & \phantom{u_4} & \smash{\vdots} & \\
u_n &-& u_{n-1} &=& r \\
\phantom{u_5} & & \phantom{u_4} & \smash{\vdots} & \\
u_n &-& u_0 &=& nr
\end{align*}
\CodeAfter
\begin{NiceMatrix}
\tikz \draw [very thick, red, opacity=0.4, name suffix = -medium] \\
\draw (1-1.north west) -- (2-3.south east) \\
(2-1.north west) -- (3-3.south east) \\
(3-1.north west) -- (4-3.south east) \\
(4-1.north west) -- (5-3.south east) \\
(5-1.north west) -- (6-3.south east) ;
\end{NiceMatrix}

15.3 The nodes which indicate the position of the rules

The package `nicematrix` creates a PGF/Tikz node merely called \( i \) (with the classical prefix) at the intersection of the horizontal rule of number \( i \) and the vertical rule of number \( i \) (more specifically the potential position of those rules because maybe there are not actually drawn). The last node has also an alias called `last`. There is also a node called \( i.5 \) midway between the node \( i \) and the node \( i + 1 \).

These nodes are available in the `CodeBefore` and the `CodeAfter`.

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

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 `CodeBefore`, to the intersection of the (potential) horizontal rule \( i \) and the (potential) vertical rule \( j \) with the syntax \((i-1j)\).

\begin{NiceMatrix}
\begin{tikz} \draw [fill=red!15] (7-4) |- (8-15) |- (9-16) |- cycle ; \end{tikz}
\end{NiceMatrix}
The nodes of the form $i.5$ may be used, for example to cross a row of a matrix (if Tikz is loaded).

$$\begin{pNiceArray}{ccc|c}
2 & 1 & 3 & 0 \\
3 & 3 & 1 & 0 \\
3 & 3 & 1 & 0
\CodeAfter
\tikz \draw [red] (3.5|-1) -- (3.5|-last) ;
\end{pNiceArray}$$

$$\begin{pmatrix}
121 & 23 & 345 & 345 \\
45 & \{346 & 863 & 444 \} \\
3462 & \{38458 & 34 & 294 \}
\end{pmatrix}$$

15.4 The nodes corresponding to the command $\text{\textbackslash SubMatrix}$

The command $\text{\textbackslash SubMatrix}$ available in the $\text{\textbackslash CodeAfter}$ has been described p. 29.

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

The nodes $\text{MyName-left}$ and $\text{MyName-right}$ correspond to the delimiters left and right and the node $\text{MyName}$ correspond to the submatrix itself.

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

16 API for the developpers

The package nicematrix provides two variables which are internal but public:

- $\text{$\backslash g\_nicematrix\_code\_before\_tl}$;
- $\text{$\backslash g\_nicematrix\_code\_after\_tl}$.

These variables contain the code of what we have called the “code-before” (usually specified at the beginning of the environment with the syntax using the keywords $\text{$\backslash CodeBefore$}$ and $\text{$\backslash Body$}$) and the “code-after” (usually specified at the end of the environment after the keyword $\text{$\backslash CodeAfter$}$). The developper 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 $\text{$\backslash 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).

\footnote{According to the LaTeX3 conventions, each variable with name beginning with $\text{$g\_nicematrix$}$ or $\text{$l\_nicematrix$}$ is public and each variable with name beginning with $\text{$g\_\_nicematrix$}$ or $\text{$l\_\_nicematrix$}$ is private.}
Example: We want to write a command \texttt{\crossbox} to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs \textit{key}-\textit{value} which will be given to Tikz before the drawing. It’s possible to program such command \texttt{\crossbox} as follows, explicitly using the public variable \texttt{\g_nicematrix_code_after_tl}.

\begin{verbatim}
\ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_crossbox:nnn { \tikz \draw [ #3 ]
( #1 -| \int_eval:n { #2 + 1 } ) -- ( \int_eval:n { #1 + 1 } -| #2 )
( #1 -| #2 ) -- ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ; }
\NewDocumentCommand \crossbox { ! O { } } { \tl_gput_right:Nx \g_nicematrix_code_after_tl { \__pantigny_crossbox:nnn { \int_use:c { c@iRow } } { \int_use:c { c@jCol } } { \exp_not:n { #1 } } } }
\ExplSyntaxOff
\end{verbatim}

Here is an example of utilisation:
\begin{NiceTabular}{ccc}[hvlines]
merlan & requin & cabillaud \\
baleine \crossbox[red] & morue \\
mante & raie & poule
\end{NiceTabular}

\begin{tabular}{ccc}
merlan & requin & cabillaud \\
baleine & morue \\
mante & raie & poule
\end{tabular}

17 Technical remarks

17.1 Diagonal lines

By default, all the diagonal lines\footnote{We speak of the lines created by \texttt{\Ddots} and not the lines created by a command \texttt{\line} in the \texttt{\CodeAfter}.} 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):

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

$$A = \begin{pmatrix}
1 & \Cdots & & 1 \\
a+b & \Ddots & & \Vdots \\
\Vdots & \Ddots & & \\
a+b & \Cdots & a+b & 1
\end{pmatrix}$$
\$A = \begin{pNiceMatrix}
1 & \Cdots & & 1 \\
& a+b & & \Vdots \\
\Vdots & \Ddots & \Ddots & \\
& a+b & \Cdots & a+b & 1
\end{pNiceMatrix}
A = \begin{pmatrix}
1, & \ldots, & 1 \\
a+b & & \\
\ldots & & \ldots & & a+b & 1
\end{pmatrix}

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

The same example without parallelization:

A = \begin{pmatrix}
1, & \ldots, & 1 \\
a+b & & \\
\ldots & & \ldots & & a+b & 1
\end{pmatrix}

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

\[\Ddots[\text{draw-first}]\]

17.2 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. When the key `corners` is used (cf. p. 10), `nicematrix` computes corners consisting of empty cells. 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 `&`). The precise rules are as follow.

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

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

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

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

- A cell 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`.

- A cell of a column of type `p`, `m` or `t` is always considered as not empty. \textit{Caution} : One should not rely upon that point because it may change in a future version of `nicematrix`. On the other side, a cell of a column of type `V` of `varwidth` (cf. p. 20) is empty when its TeX content has a width equal to zero.

17.3 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\footnote{In the documentation of `{amsmath}`, we can read: \textit{The extra space of \texttt{arraycolsep} that \texttt{array} adds on each side is a waste so we remove it \texttt{[in \texttt{matrix}]} (perhaps we should instead remove it from array in general, but that’s a harder task).}}. The environment `{matrix}`
of \texttt{amsmath} and its variants (\{\texttt{pmatrix}, \texttt{vmatrix}\}, etc.) of \texttt{amsmath} prefer to delete these spaces with explicit instructions \texttt{\backslash hskip -\arraycolsep}\cite{footnote:12}. The package \texttt{nicematrix} does the same in all its environments, \{\texttt{NiceArray}\} included. However, if the user wants the environment \{\texttt{NiceArray}\} behaving by default like the environment \{\texttt{array}\} of \texttt{array} (for example, when adapting an existing document) it’s possible to control this behaviour with the option \texttt{exterior-arraycolsep}, set by the command \texttt{\NiceMatrixOptions}. With this option, exterior spaces of length \texttt{\arraycolsep} will be inserted in the environments \{\texttt{NiceArray}\} (the other environments of \texttt{nicematrix} are not affected).

### 17.4 Incompatibilities

The package \texttt{nicematrix} is not compatible with the class \texttt{ieeeaccess} (because that class is not compatible with PGF/Tikz).\cite{footnote:13}

In order to use \texttt{nicematrix} with the class \texttt{aastex631}, you have to add the following lines in the preamble of your document:

\begin{verbatim}
\BeforeBegin{NiceTabular}{\let\begin\BeginEnvironment\let\end\EndEnvironment}
\BeforeBegin{NiceArray}{\let\begin\BeginEnvironment}
\BeforeBegin{NiceMatrix}{\let\begin\BeginEnvironment}
\end{verbatim}

In order to use \texttt{nicematrix} with the class \texttt{sn-jnl}, \texttt{pgf} must be loaded before the \texttt{documentclass}:

\begin{verbatim}
\RequirePackage{pgf}
\documentclass{sn-jnl}
\end{verbatim}

The package \texttt{nicematrix} is not fully compatible with the package \texttt{arydshln} (because this package redefines many internal of \texttt{array}). By any means, in the context of \texttt{nicematrix}, it’s recommended to draw dashed rules with the tools provided by \texttt{nicematrix}, by creating a customized line style with \texttt{custom-line}: cf. p. 11.

### 18 Examples

#### 18.1 Utilisation of the key “tikz” of the command \texttt{\Block}

The key \texttt{tikz} of the command \texttt{\Block} is available only when Tikz is loaded.\cite{footnote:14}

For the following example, we need also the Tikz library \texttt{patterns}.

\begin{verbatim}
\usetikzlibrary{patterns}
\ttfamily \small
\begin{NiceTabular}{X[m]X[m]X[m]}[hvlines,cell-space-limits=3pt]
  \Block[tikz={pattern=grid,pattern color=lightgray}]{}
  {pattern = grid,\ \ pattern color = lightgray}
  \& \Block[tikz={pattern = north west lines,pattern color=blue}]{}
  {pattern = north west lines,\ \ pattern color = blue}
  \& \Block[tikz={outer color = red!50, inner color=white }]\{2-1}
  {outer color = red!50,\ \ inner color = white} \ \
  \Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{}
  {pattern = sixpointed stars,\ \ pattern color = blue!15}
\end{NiceTabular}
\end{verbatim}

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

\footnote{See \url{https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class}}

\footnote{By default, \texttt{nicematrix} only loads \texttt{pgf}, which is a sub-layer of Tikz.}
18.2 Notes in the tabulars

The tools provided by nicematrix for the composition of the tabular notes have been presented in the section 12 p. 32.

Let’s consider that we wish to number the notes of a tabular with stars. 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.

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

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}
\toprule \RowStyle{\bfseries}
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 \\
\end{NiceTabular}

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

\footnote{In fact: the value of its argument.}
Achard* Jacques June 5, 2005
Lefebvre** Mathilde January 23, 1975
Vanesse Stephany October 30, 1994
Dupont Chantal January 15, 1998

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

18.3 Dotted lines

An example with the resultant of two polynomials:

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

An example for a linear system:

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

\begin{verbatim}
\NiceMatrixOptions{nullify-dots,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\text{-}line-style={dashed,\blue}]
1 & & & \Vdots & & & & \Vdots \\
 & \Ddots[\text{-}line-style=standard] \\
 & & 1 \\
\Cdots[\text{-}color=\blue,\text{-}line-style=dashed]& & & \blue 0 & \Cdots & & & \blue 1 & & & \Cdots & \blue \leftarrow i \\
 & & & & 1 \\
 & & &\Vdots & & \Ddots[\text{-}line-style=standard] & & \Vdots \\
 & & & & & & 1 \\
\Cdots & & & \blue 1 & \Cdots & & \Cdots & \blue 0 & & & \Cdots & \blue \leftarrow j \\
\end{pNiceMatrix}
\end{verbatim}

\begin{center}
\begin{pmatrix}
1 & 1 & \ldots & 1 \\
0 & 1 & \ldots & 0 \\
0 & 0 & 1 & \ldots \\
0 & \ldots & 0 & 1
\end{pmatrix}
\begin{pmatrix}
L_2 \leftarrow L_2 - L_1 \\
L_3 \leftarrow L_3 - L_1 \\
\vdots \\
L_n \leftarrow L_n - L_1
\end{pmatrix}
\end{center}

\begin{center}
\begin{pmatrix}
1 & & & & & & & \\
& & & & & & \leftarrow i \\
& & & & & & \leftarrow j \\
& & & & & & \downarrow i \\
& & & & & & \downarrow j
\end{pmatrix}
\end{center}

In fact, it’s even possible to draw solid lines with the commands \texttt{\cdots}, \texttt{\vdots}, etc.\textsuperscript{56}

\begin{verbatim}
\NiceMatrixOptions{nullify-dots,code-for-first-row = \color{blue},code-for-first-col = \color{blue}}
\begin{pNiceMatrix}[first-row,first-col]
\& & \Ldots[\text{-}line-style={solid,\leftarrow},\text{-}shorten=0pt]^{n \text{ \text{\scriptsize columns}}} \\\n\& 1 & 1 & 1 & \Ldots & 1 \\
\& 1 & 1 & 1 & \Ldots & 1 \\
\& 1 & 1 & 1 & \Ldots & 1 \\
\& 1 & 1 & 1 & \Ldots & 1 \\
\& 1 & 1 & 1 & \Ldots & 1 \\
\end{pNiceMatrix}
\end{verbatim}

\textsuperscript{56}In this document, the Tikz library \texttt{arrows.meta} has been loaded, which impacts the shape of the arrow tips.
\begin{NiceMatrix}
\Block[borders={bottom,right,tikz=dashed}]{2-2}{1 & 2 & 0 & 0 & 0 & 0 \\
4 & 5 & 0 & 0 & 0 & 0 \}
0 & 0 & \Block[borders={bottom,top,right,left,tikz=dashed}]{2-2}{7 & 1 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 \}
0 & 0 & 0 & 0 & \Block[borders={left,top,tikz=dashed}]{2-2}{3 & 4 \}
0 & 0 & 0 & 0 & 1 & 4
\end{NiceMatrix}

18.5 Dashed rules

In the following example, we use the command \texttt{Block} to draw dashed rules. For that example, Tikz should be loaded (by \texttt{usepackage{tikz}}).

\begin{pNiceMatrix}
\Block[borders={bottom,right,tikz=dashed}]{2-2}{} \\
1 & 2 & 0 & 0 & 0 & 0 \\
4 & 5 & 0 & 0 & 0 & 0 \\
0 & 0 & \Block[borders={bottom,top,right,left,tikz=dashed}]{2-2}{} \\
7 & 1 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & \Block[borders={left,top,tikz=dashed}]{2-2}{} \\
3 & 4 \\
0 & 0 & 0 & 0 & 1 & 4
\end{pNiceMatrix}

18.6 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{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}\$
\[\begin{pNiceMatrix}{rrrr|r}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \{ L_2 \text{ gets } L_1-4L_2 \} \\
0 & -192 & 123 & -3 & -57 \{ L_3 \text{ gets } L_1+4L_3 \} \\
0 & -64 & 41 & -1 & -19 \{ L_4 \text{ gets } 3L_1-4L_4 \}
\end{pNiceMatrix}\]

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

\[\begin{pNiceMatrix}{rrrr|r}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0
\end{pNiceMatrix}\]

\begin{NiceMatrixBlock}[]
\begin{NiceMatrixOptions}
  \delimiters/max-width, \\
  light-syntax, \\
  last-col, code-for-last-col = \color{blue}\scriptstyle,
\end{NiceMatrixOptions}
\setlength{\extrarowheight}{1mm}
\begin{pNiceMatrix}{rrrr|r}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7
\end{pNiceMatrix}
\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 \text{ gets } L_1-4L_2 \} \\
0 & -192 & 123 & -3 & -57 \{ L_3 \text{ gets } L_1+4L_3 \} \\
0 & -64 & 41 & -1 & -19 \{ L_4 \text{ gets } 3L_1-4L_4 \}
\end{pmatrix}
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19
\end{pmatrix}
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 0 & 0 & 0 & 0 \{ L_3 \text{ gets } 3L_2 + L_3 \}
\end{pmatrix}
\]

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

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 \texttt{delimiters/max-width}.

\begin{NiceMatrixBlock}[\texttt{auto-columns-width}]
\NiceMatrixOptions
\setlength{\extrarowheight}{1mm}
\begin{pNiceMatrix}{rrrr|r}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7
\end{pNiceMatrix}
\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 \text{ gets } L_1-4L_2 \} \\
0 & -192 & 123 & -3 & -57 \{ L_3 \text{ gets } L_1+4L_3 \} \\
0 & -64 & 41 & -1 & -19 \{ L_4 \text{ gets } 3L_1-4L_4 \}
\end{pmatrix}
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19
\end{pmatrix}
\begin{pmatrix}
12 & -8 & 7 & 5 & 3 \\
0 & 0 & 0 & 0 & 0 \{ L_3 \text{ gets } 3L_2 + L_3 \}
\end{pmatrix}
\]

\end{NiceMatrixBlock}
\end{pNiceArray}

\[...
\end{NiceMatrixBlock}

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\
\end{array}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & -192 & 123 & -3 & -57 \\
0 & -64 & 41 & -1 & -19 \\
\end{array}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\end{NiceMatrix}
\]

\[
\begin{NiceMatrix}
\begin{array}{cccc|c}
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\end{NiceMatrix}
\]

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 \SubMatrix in the \CodeAfter. Of course, that array can’t be broken by a page break.

\setlength{\extrarowheight}{1mm}
\[
\begin{NiceMatrix}[ r, last-col=6, code-for-last-col = \scriptstyle \color{blue} ]
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\
0 & 64 & -41 & 1 & 19 \text{ } \text{ } L_2 \gets L_1-4L_2 \\
0 & -192 & 123 & -3 & -57 \text{ } \text{ } L_3 \gets L_1+4L_3 \\
0 & -64 & 41 & -1 & -19 \text{ } \text{ } L_4 \gets L_1-4L_4 \\
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \text{ } \text{ } L_3 \gets 3L_2+L_3 \\
0 & 12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \text{ } \text{ } L_3 \gets 3L_2+L_3 \\
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \text{ } \text{ } L_3 \gets 3L_2+L_3 \\
0 & 12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
\end{NiceMatrix}
\]

\begin{CodeAfter} [sub-matrix/vlines=4]
\SubMatrix{(1-1){4-5}}
\SubMatrix{(5-1){8-5}}
\SubMatrix{(9-1){11-5}}
\SubMatrix{(12-1){13-5}}
\end{NiceMatrix}\]

53
In this tabular, the instructions \SubMatrix are executed after the composition of the tabular and, thus, the vertical rules are drawn without adding space between the columns.

In fact, it’s possible, with the key \texttt{vlines-in-sub-matrix}, to choice a letter in the preamble of the array to specify vertical rules which will be drawn in the \SubMatrix only (by adding space between the columns).

\setlength{\extrarowheight}{1mm}
\begin{NiceArray}
\begin{array}{rrrrIr}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\
0 & 64 & -41 & 1 & 19 \[L_2=\!L_1\!-\!4L_2\] \\
0 & -192 & 123 & -3 & -57 \[L_3=\!L_1\!+\!4L_3\] \\
0 & -64 & 41 & -1 & -19 \[L_4=\!3L_1\!-\!4L_4\] \\
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \[L_3=\!3L_2\!+L_3\] \\
0 & 0 & 0 & 0 & 0 \[L_4=3L_1\!-\!4L_4\] \\
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \[L_3=\!3L_2\!+L_3\] \\
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 \[L_3=\!3L_2\!+L_3\] \\
\end{array}
\end{NiceArray}\[54]
18.7 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\footnote{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}).

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

\begin{pNiceArray}{>{\strut}ccc}
\rowcolor{red!15} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \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 commands \texttt{\hline} and \texttt{\Hline}, the specifier “|” and the options \texttt{hlines}, \texttt{vlines}, \texttt{hvlines} and \texttt{hvlines-except-borders} spread the cells.\footnote{For the command \texttt{\cline}, see the remark p. 8.}

It’s possible to color a row with \texttt{\rowcolor} in the \texttt{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}ccc}
\rowcolor{red!15} A_{11} & A_{12} & A_{13} \\ A_{21} & \rowcolor{red!15} A_{22} & A_{23} \\ A_{31} & A_{32} & \rowcolor{red!15} A_{33} \\ A_{41} & A_{42} & A_{43} \end{pNiceArray}

57We 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.

58For the command \texttt{\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.

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

\begin{verbatim}
\usepackage{tikz}
\usetikzlibrary{fit}
\end{verbatim}

We create a rectangular Tikz node which encompasses the nodes of the second row by using the tools of the Tikz library fit. Those nodes are not available by default in the \texttt{CodeBefore} (for efficiency). We have to require their creation with the key \texttt{create-cell-nodes} of the keyword \texttt{CodeBefore}.

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

\begin{verbatim}
\begin{bNiceMatrix}
[create-cell-nodes]
\tikz \node [highlight = (2-1) (2-3)] {} ;
\end{bNiceMatrix}
\end{verbatim}

\begin{verbatim}
\begin{pNiceArray}{ccc}[last-col]
[create-cell-nodes]
\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\end{pNiceArray}
\end{verbatim}

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

\begin{verbatim}
\begin{pNiceArray}{ccc}[last-col]
\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\end{pNiceArray}
\end{verbatim}

\begin{verbatim}
\begin{pmatrix}
0 & \ldots & 0 \\
1 & \ldots & 1 \\
0 & \ldots & 0
\end{pmatrix}
\end{verbatim}

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

\begin{verbatim}
\begin{pmatrix}
0 & \ldots & 0 \\
1 & \ldots & 1 \\
0 & \ldots & 0
\end{pmatrix}
\end{verbatim}

\section*{56}
The result may seem disappointing. We can improve it by using the “medium nodes” instead of the “normal nodes”.

\[
\begin{pNiceArray}{ccc}
[\text{last-col,create-medium-nodes}]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture}
\[\text{name suffix = -medium}\]
\node \[\text{highlight = (1-1) (1-3)}\] {} ;
\node \[\text{highlight = (2-1) (2-3)}\] {} ;
\node \[\text{highlight = (3-1) (3-3)}\] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
\text{nullify-dots}
\text{SubMatrix}({2-7}{6-11})
\text{SubMatrix}({7-2}{11-6})
\text{SubMatrix}({7-7}{11-11})
\begin{tikzpicture}
\tikzset{highlight/.style={rectangle,
fill=red!15,
rounded corners = 0.5 mm,
inner sep=1pt,
fit=#1}}
\end{tikzpicture}
\end{NiceArray}
\]

18.8 Utilisation of \texttt{\textbackslash SubMatrix} in the \texttt{\textbackslash CodeBefore}

In the following example, we illustrate the mathematical product of two matrices. The whole figure is an environment \texttt{\textbackslash NiceArray} and the three pairs of parenthesis have been added with \texttt{\textbackslash SubMatrix} in the \texttt{\textbackslash CodeBefore}.
\begin{NiceArray}{ccccccccc}
\CodeAfter
& & & & & & & & \color{blue}\scriptstyle C_j \\
& & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
& & \Vdots & & \Vdots & & \Vdots & & \Vdots \\
& & & & & b_{kj} \\
& & & & & \Vdots \\
& & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\
& a_{11} & \Cdots & & & a_{1n} \\
\CodeAfter
\end{NiceArray}

\begin{tikzpicture}
\node [highlight = (9-2) (9-6)] { } ;
\node [highlight = (2-9) (6-9)] { } ;
\end{tikzpicture}

19 Implementation

By default, the package \texttt{nicematrix} doesn't patch any existing code.

However, when the option \texttt{renew-dots} is used, the commands \texttt{\cdots}, \texttt{\ldots}, \texttt{\dots}, \texttt{\vdots}, \texttt{\ddots} and \texttt{\iddots} are redefined in the environments provided by \texttt{nicematrix} as explained previously.

In the same way, if the option \texttt{renew-matrix} is used, the environment \texttt{\{matrix\}} of \texttt{amsmath} is redefined.

On the other hand, the environment \texttt{\{array\}} is never redefined.

Of course, the package \texttt{nicematrix} uses the features of the package \texttt{array}. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent. For example, the package \texttt{nicematrix} relies upon the fact that the package \texttt{\{array\}} uses \texttt{\ialign} to begin the \texttt{\halign}.

Declaration of the package and packages loaded

The prefix \texttt{nicematrix} has been registered for this package.

See: \url{http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf}
<@@=nicematrix>

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

1 \RequirePackage{pgfcore}
2 \usepgfmodule{shapes}

We give the traditional declaration of a package written with the L3 programming layer.

3 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5 \nicematrix
6 \\myfiledate
7 \\myfileversion
8 \{Enhanced arrays with the help of PGF/TikZ\}

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

We load some packages.

9 \RequirePackage { \{ array \} }
10 \RequirePackage { amsmath }

58
With Overleaf, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key H in order to have more information. That’s why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key messages-for-Overleaf is used (at load-time).

Technical definitions

In some constructions, we will have to use a \texttt{pgfpicture} which \textit{must} be replaced by a \texttt{tikzpicture} if Tikz is loaded. However, this switch between \texttt{pgfpicture} and \texttt{tikzpicture} can’t be done dynamically with a conditional because, when the Tikz library \texttt{external} is loaded by the user, the pair \texttt{\begin{tikzpicture}-\end{tikzpicture}} (or \texttt{\begin{tikzpicture}}-\texttt{\end{tikzpicture}}) must be statically “visible” (even when externalization is not activated). That’s why we create \texttt{\c_@@_pgfortikzpicture_tl} and \texttt{\c_@@_endpgfortikzpicture_tl} which will be used to construct in a \texttt{\AtBeginDocument} the correct version of some commands. The tokens \texttt{\exp_not:N} are mandatory.
We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefine `\array` (of `array`) in a way incompatible with our programmation. At the date January 2022, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```latex
\ifsclassloaded { revtex4-1 }
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
{ \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
\fi
```

Maybe one of the previous classes will be loaded inside another class... We try to detect that situation.

```latex
\cs_if_exist:NT \rvtx@ifformat@geq
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
{ \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
```

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

```latex
\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.

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

We define a command `\iddots` similar to `\ddots` (```) but with dots going forward (```). 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.

```latex
\ProvideDocumentCommand \iddots { }
{ \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
}```
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.

\hook_gput_code:nnn { begindocument } { . }
\{ 
  \@ifpackageloaded { booktabs } 
  { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun } 
  { } 
\} 
\cs_set_protected:Npn \nicematrix@redefine@check@rerun 
\{ 
  \cs_set_eq:NN \@@_old_pgfutil@check@rerun \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).

\hook_gput_code:nnn { begindocument } { . }
\{ 
  \@ifpackageloaded { booktabs } 
  { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun } 
  { } 
\} 
\cs_set_protected:Npn \nicematrix@redefine@check@rerun 
\{ 
  \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun
\} 

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

\bool_new:N \l_@@_colortbl_loaded_bool
\hook_gput_code:nnn { begindocument } { . }
\{ 
  \@ifpackageloaded { colortbl } 
  { \bool_set_true:N \l_@@_colortbl_loaded_bool } 
  { } 
\}

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

\cs_set:Npn \CT@arc@ { } 
\cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } } 
\cs_set:Npn \CT@arc #1 #2 
\{ 
  \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign 
  { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } } 
\}

Idem for $\CT@drs@$.

\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@drs@ { \color #1 { #2 } } } 
\}
\cs_set:Npn \hline 
\{ 
  \noalign { \ifnum 0 = `} \fi 
  \cs_set_eq:NN \haskip \vskip 
  \cs_set_eq:NN \vrule \hrule 
  \cs_set_eq:NN \@width \@height 
  \futurelet \reserved@a \hline 
\}
\}
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.

\begin{verbatim}
\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 { \int_eval:n { #1 - 1 } } } &
\multispan { \int_eval:n { #2 - #1 + 1 } }
\}
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim
154
\everycr
155
\cr
156
\noalign { \skip_vertical:N -\arrayrulewidth }
\}
\cs_set:Npn \@@_cline
\everycr { }
\cr

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 \everycr { }
\cr

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).

\begin{verbatim}
\int_compare:nNnT \#1 < \#2
\{ \multispan \{ \int_eval:n \{ \#2 - \#1 \} \} &
\multispan \{ \int_eval:n \{ \#3 - \#2 + 1 \} \}
\}
\CT@arc@
\leaders \hrule \@height \arrayrulewidth \hfill
\skip_horizontal:N \c_zero_dim
\}
\peek_meaning_remove_ignore_spaces:NTF \cline
\{ \everycr { } \cr
\}
\end{verbatim}

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

\begin{verbatim}
\peek_meaning_remove_ignore_spaces:NTF \cline
\{ \everycr { } \cr
\}
\end{verbatim}

\end{verbatim}

59\textsuperscript{See question 99041 on TeX StackExchange.}
The following command is a small shortcut.
\cs_generate_variant:Nn \@@_cline_i:nn { e n }

The command \@@_renew_NC@rewrite@S: will be used in each environment of \nicematrix in order to "rewrite" the $S$ column in each environment.

Parameters

The following counter will count the environments \{NiceArray\}. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.
The following command is only a syntactic shortcut. It must not be protected (it will be used in names of PGF nodes).
\cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }

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

The following command is only a syntactic shortcut. The q in qpoint means quick.
\cs_new_protected:Npn \@@_qpoint:n #1
\{ \pgfpointanchor{\@@_env: - #1}{center} \}

The following counter will count the environments {NiceMatrixBlock}.
\int_new:N \g_@@_NiceMatrixBlock_int

The dimension \l_@@_columns_width_dim will be used when the options specify that all the columns must have the same width (but, if the key columns-width is used with the special value auto, the boolean \l_@@_auto_columns_width_bool also will be raised).
\dim_new:N \l_@@_columns_width_dim
\dim_set:Nn \l_@@_columns_width_dim { -1 cm }

The dimension \l_@@_col_width_dim will be available in each cell which belongs to a column of fixed width: \w{...}{...}, \W{...}{...}, \p{...}{...}, \m{...}{...}, \b{...}{...} but also X (when the actual width of that column is known, that is to say after the first compilation). It’s the width of that column. It will be used by some commands \Block. A non positive value means that the column has no fixed width (it’s a column of type c, r, l, etc.).
\dim_new:N \l_@@_col_width_dim
\dim_set:Nn \l_@@_col_width_dim { -1 cm }

The following counters will be used to count the numbers of rows and columns of the array.
\int_new:N \g_@@_row_total_int
\int_new:N \g_@@_col_total_int

The following parameter will be used by \@@_create_row_node: to avoid to create the same row-node twice (at the end of the array).
\int_new:N \g_@@_last_row_node_int

The following counter corresponds to the key nb-rows of the command \RowStyle.
\int_new:N \l_@@_key_nb_rows_int

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are r, l, c. For example, a column p[l]{3cm} will provide the value l for all the cells of the column.
\str_new:N \l_@@_hpos_cell_str
\str_set:Nn \l_@@_hpos_cell_str { c }

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

240 \dim_new:N \g_@@_blocks_ht_dim
241 \dim_new:N \g_@@_blocks_dp_dim

The following dimension correspond to the key width (which may be fixed in \NiceMatrixOptions but also in an environment \NiceTabular).

242 \dim_new:N \l_@@_width_dim

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

243 \seq_new:N \g_@@_names_seq

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

244 \bool_new:N \l_@@_in_env_bool

The following key corresponds to the key notes/detect_duplicates.

245 \bool_new:N \l_@@_notes_detect_duplicates_bool
246 \bool_set_true:N \l_@@_notes_detect_duplicates_bool

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

247 \bool_new:N \l_@@_NiceArray_bool

In fact, if there is delimiters in the preamble of \{NiceArray\} (eg: \[cccc\]), this boolean will be set to false.

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

248 \bool_new:N \l_@@_NiceTabular_bool

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

249 \dim_new:N \l_@@_tabular_width_dim

The following dimension will be used for the total width of composite rules (total means that the spaces on both sides are included).

250 \dim_new:N \l_@@_rule_width_dim

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

251 \bool_new:N \l_@@_Matrix_bool

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

252 \bool_new:N \g_@@_rotate_bool

In a cell, it will be possible to know whether we are in a cell of a column of type X thanks to that flag.

253 \bool_new:N \l_@@_X_column_bool

We will write in \g_@@_aux_tl all the instructions that we have to write on the aux file for the current environment. The contain of that token list will be written on the aux file at the end of the environment (in an instruction \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }).

254 \tl_new:N \g_@@_aux_tl
The letter used for the vlines which will be drawn only in the sub-matrices. vlism stands for _vertical lines in sub-matrices._

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.

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

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

The following string will contain the word _command_ or _environment_ whether we are in a command of nicematrix or in an environment of nicematrix. The default value is _environment_.

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

The following token list corresponds to the option _code-after_ (it's also possible to set the value of that parameter with the keyword _CodeAfter_). That parameter is _public._

For the key _code_ of the command `\SubMatrix` (itself in the main _CodeAfter_), we will use the following token list.

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

\begin{verbatim}
\int_new:N \l_@@_old_iRow_int
\int_new:N \l_@@_old_jCol_int
\end{verbatim}

The TeX counters \texttt{\_\_\_old_iRow} and \texttt{\_\_\_old_jCol} will be created in the beginning of \texttt{\texttt{NiceArrayWithDelims}} (if they don’t exist previously).

The following sequence will contain the names (without backslash) of the commands created by \texttt{custom-line} (commands used by the final user in order to draw horizontal rules).

\begin{verbatim}
\seq_new:N \l_@@_custom_line_commands_seq
\end{verbatim}

The following token list corresponds to the key \texttt{rules/color} available in the environments.

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

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as optional argument between square brackets. The default value, of course, is 1.

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

If there is at least one X-column in the preamble of the array, the following flag will be raised via the aux file. The length $\texttt{\_\_\_X_columns_dim}$ will be the width of X-columns of weight 1 (the width of a column of weight $n$ will be that dimension multiplied by $n$). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

\begin{verbatim}
\bool_new:N \l_@@_X_columns_aux_bool
\dim_new:N \l_@@_X_columns_dim
\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{\_\_\_dotsfor} is used in that column.

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

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

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

\texttt{\_\_\_code_before_tl} may contain two types of informations:

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

- The final user can explicitly add material in $\texttt{\_\_\_code_before_tl}$ 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 token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

\begin{verbatim}
\tl_new:N \g_@@_row_style_tl
\end{verbatim}
The following dimensions will be used when drawing the dotted lines.
\dim_new:N \l_@@_x_initial_dim
\dim_new:N \l_@@_y_initial_dim
\dim_new:N \l_@@_x_final_dim
\dim_new:N \l_@@_y_final_dim

The L3 programming layer provides scratch dimensions $\l_{tmpa}_dim$ and $\l_{tmpb}_dim$. We create two more in the same spirit.
\dim_zero_new:N \l_@@_tmpc_dim
\dim_zero_new:N \l_@@_tmpd_dim

Some cells will be declared as “empty” (for example a cell with an instruction $\Cdots$).
\bool_new:N \g_@@_empty_cell_bool

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.
\dim_new:N \g_@@_width_last_col_dim
\dim_new:N \g_@@_width_first_col_dim

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

We also manage a sequence of the positions of the blocks. In that sequence, each block is represented by only five components: \{imin\},\{imin\},\{imax\},\{imax\},\{name\}. A block with the key hvlines won’t appear in that sequence (otherwise, the lines in that block would not be drawn!).
\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 five components: \{imin\},\{imin\},\{imax\},\{imax\},\{name\}.
\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

If the user has used the key corners, all the cells which are in an (empty) corner will be stored in the following sequence.
\seq_new:N \l_@@_corners_cells_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 flag will be raised if the key width is used in an environment \NiceTabular (not in a command \NiceMatrixOptions). You use it to raise an error when this key is used while no column X is used.
\bool_new:N \l_@@_width_used_bool
The sequence \texttt{\g_@@_multicolumn_cells_seq} will contain the list of the cells of the array where a command \texttt{\multicolumn\{n\}{...}{...}} with \(n > 1\) is issued. In \texttt{\g_@@_multicolumn_sizes_seq}, the “sizes” (that is to say the values of \(n\)) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

\begin{verbatim}
\seq_new:N \g_@@_multicolumn_cells_seq
\seq_new:N \g_@@_multicolumn_sizes_seq
\end{verbatim}

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 \texttt{\SubMatrix}—the \texttt{\SubMatrix} in the \texttt{code-before}).

\begin{verbatim}
\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
\end{verbatim}

The following sequence will be used when the command \texttt{\SubMatrix} is used in the \texttt{CodeBefore} (and not in the \texttt{CodeAfter}). It will contain the position of all the sub-matrices specified in the \texttt{CodeBefore}. Each sub-matrix is represented by an “object” of the forme \texttt{\{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.

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

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).

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

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

\begin{verbatim}
\tl_new:N \l_@@_fill_tl
\tl_new:N \l_@@_draw_tl
\seq_new:N \l_@@_tikz_seq
\clist_new:N \l_@@_borders_clist
\dim_new:N \l_@@_rounded_corners_dim
\end{verbatim}

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by \texttt{nicematrix} when the key \texttt{corners} is used).

The following token list correspond to the key \texttt{color} of the command \texttt{\Block}.

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

Here is the dimension for the width of the rule when a block (created by \texttt{\Block}) is stroked.

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

The parameters of the horizontal position of the label of a block. If the user uses the key \texttt{c} or \texttt{C}, the value is \texttt{c}. If the user uses the key \texttt{l} or \texttt{L}, the value is \texttt{l}. If the user uses the key \texttt{r} or \texttt{R}, the value is \texttt{r}. If the user has used a capital letter, the boolean \texttt{\l_@@_hpos_of_block_cap_bool} will be raised (in the second pass of the analyze of the keys of the command \texttt{\Block}).

\begin{verbatim}
\str_new:N \l_@@_hpos_block_str
\str_set:Nn \l_@@_hpos_block_str { \texttt{c} }
\bool_new:N \l_@@_hpos_of_block_cap_bool
\end{verbatim}

For the vertical position, the possible values are \texttt{c}, \texttt{t} and \texttt{b}. Of course, it would be interesting to program a key \texttt{T} and a key \texttt{B}.

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

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

\begin{verbatim}
\bool_new:N \l_@@_draw_first_bool
\end{verbatim}
The following flag corresponds to the keys vlines and hlines of the command \Block (the key hvlines is the conjunction of both).

\bool_new:N \l_@@_vlines_block_bool
\bool_new:N \l_@@_hlines_block_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

The following flag will be used by (for instance) \@@_vline_ii:. When \l_@@_dotted_bool is true, a dotted line (with our system) will be drawn.

\bool_new:N \l_@@_dotted_bool

Variables for the exterior rows and columns

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

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

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

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

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

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

  \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.}
Idem for \_l_@@_last_col_without_value_bool
\bool_new:N \l_@@_last_col_without_value_bool

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

However, we have also a boolean. Consider the following code:
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}

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

This boolean is set to false at the end of \@@_pre_array_ii:.

Some utilities
\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 }
}
The following takes as argument the name of a clist and which should be a list of intervals of integers. It expands that list, that is to say, it replaces (by a sort of mapcan or flat_map) the interval by the explicit list of the integers.
\cs_new_protected:Npn \@@_expand_clist:N #1
{\clist_if_in:NnF #1 { all } {
\clist_clear:N \l_tmpa_clist
\clist_map_inline:Nn #1 \l_tmpa_clist
\l_tmpb_clist \l_tmpa_clist
\l_tmpb_clist \l_tmpa_clist
\int_step_inline:nnn { \l_TMPa_tl } { \l_TMPb_tl } { \clist_put_right:Nn \l_tmpa_clist { #1#1#1#1 } }
\tl_set_eq:NN #1 \l_tmpa_clist
}}
The command \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.

\newcounter{tabularnote}

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

\seq_new:N \g_@@_tabularnotes_seq

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.

\tl_new:N \l_@@_tabularnote_tl

\seq_new:N \l_@@_notes_labels_seq

\newcounter{nicematrix_draft}

\cs_new_protected:Npn \@@_notes_format:n #1 {
\setcounter{nicematrix_draft}{#1}
\@@_notes_style:n {nicematrix_draft}
}

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

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

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

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

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

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

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

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

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

\hook_gput_code:nnn {begindocument} { . }

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

72
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 \texttt{\textbackslash tabularnote} is available in the whole document (and not only in the environments of \texttt{nicematrix}) because we want it to be available in the caption of a \texttt{\{table\}} (before the following \texttt{\{NiceTabular\}} or \texttt{\{NiceArray\}}). That’s also the reason why the variables \texttt{\c@tabularnote} and \texttt{\g_@@_tabularnotes_seq} will be cleared at the end of the environment of \texttt{nicematrix} (and not at the beginning).

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

```latex
\NewDocumentCommand \tabularnote { m }
{ \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
{ \@@_error:n { tabularnote~forbidden } }
{ \seq_map_indexed_inline:Nn \g_@@_tabularnotes_seq
{ \tl_if_eq:nnT { #1 } { ##2 } \int_set:Nn \l_tmpa_int { ##1 } \seq_map_break: }
\int_compare:nNnTF \l_tmpa_int = 0
{ \stepcounter { tabularnote }
\seq_put_right:Nx \l_@@_notes_labels_seq
{ \@@_notes_format:n { \int_use:c { \c@tabularnote } } }
\seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
}
{ \seq_put_right:Nx \l_@@_notes_labels_seq
```

You have to see whether the argument of \texttt{\textbackslash tabularnote} has yet been used as argument of another \texttt{\textbackslash tabularnote} in the same tabular. In that case, there will be only one note (for both commands \texttt{\textbackslash tabularnote}) at the end of the tabular. We search the argument of our command \texttt{\textbackslash tabularnote} in the \texttt{\g_@@_tabularnotes_seq}. The position in the sequence will be stored in \texttt{\l_tmpa_int} (0 if the text is not in the sequence yet).
If the following token is not a \tabularnote, we have finished the sequence of successive commands \tabularnote and we have to format the labels of these tabular notes (in the array). We compose those labels in a box \l_tmpa_box because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell.

\hbox_set:Nn \l_tmpa_box

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

\@@_notes_label_in_tabular:n

\seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , }

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

\addtocounter { tabularnote } { -1 }
\refstepcounter { tabularnote }
\seq_clear:N \l_@@_notes_labels_seq
\hbox_overlap_right:n { \box_use:N \l_tmpa_box }

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

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

Command for creation of rectangle nodes

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

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

\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5

{ \begin { pgfscope }
\pgfset
{ outer~sep = \c_zero_dim ,
inner~sep = \c_zero_dim ,
minimum~size = \c_zero_dim
}
\pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
\pgfnode
{ rectangle }
{ center }
{ \vbox_to_ht:nn
{ \dim_abs:n { #5 - #3 } }
{ \vfill
}
The command $\@_pgf_rect_node:nnn$ is a variant of $\@_pgf_rect_node:nnnnn$: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.\
\begin{verbatim}
\cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3 
\begin { pgfscope } 
\pgfset {
  outer~sep = \c_zero_dim ,
  inner~sep = \c_zero_dim ,
  minimum~size = \c_zero_dim
}
\pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
\pgfpointdiff { #3 } { #2 }
\pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
\pgfnode 
  { rectangle }
  { center }
  { \vbox_to_ht:nn \dim_abs:n \l_tmpb_dim }
  { \vfill \hbox_to_wd:nn \dim_abs:n \l_tmpa_dim }
\end { pgfscope }
\end{verbatim}

The options

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

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

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

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

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

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

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

\begin{verbatim}
\dim_new:N \l_@@_xdots_inter_dim
\hook_gput_code:nnn { begindocument } { . }
\dim_set:Nn \l_@@_xdots_inter_dim { 0.45 \text{ em} }
\end{verbatim}
We use a hook only by security in case \texttt{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_start_dim
\dim_new:N \l_@@_xdots_shorten_end_dim
\hook_gput_code:nnn { begindocument } { . }
{ \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 \text{ em} } }
{ \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 \text{ em} } }
```

We use a hook only by security in case \texttt{revtex4-1} is used (even though it is obsolete).

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

```
\dim_new:N \l_@@_xdots_radius_dim
\hook_gput_code:nnn { begindocument } { . }
{ \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 \text{ pt} } }
```

We use a hook only by security in case \texttt{revtex4-1} is used (even though it is obsolete).

The token list \texttt{\l_@@_xdots_line_style_tl} corresponds to the option \texttt{tikz} of the commands \texttt{\Cdots}, \texttt{\Ldots}, etc. and of the options \texttt{line-style} for the environments and \texttt{\NiceMatrixOptions}. The constant \texttt{\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 \texttt{\l_@@_light_syntax_bool} corresponds to the option \texttt{light-syntax}.

```
\bool_new:N \l_@@_light_syntax_bool
```

The string \texttt{\l_@@_baseline_tl} may contain one of the three values \texttt{t}, \texttt{c} or \texttt{b} as in the option of the environment \texttt{\{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 \texttt{\l_@@_exterior_arraycolsep_bool} corresponds to the option \texttt{exterior-arraycolsep}. If this option is set, a space equal to \texttt{\arraycolsep} will be put on both sides of an environment \texttt{\{NiceArray\}} (as it is done in \texttt{\{array\}} of \texttt{\{array\}}).

```
\bool_new:N \l_@@_exterior_arraycolsep_bool
```

The flag \texttt{\l_@@_parallelize_diags bool} controls whether the diagonals are parallelized. The initial value is \texttt{true}.

```
\bool_new:N \l_@@_parallelize_diags bool
\bool_set_true:N \l_@@_parallelize_diags bool
```

The following parameter correspond to the key \texttt{corners}. The elements of that \texttt{clist} must be in \texttt{NW}, \texttt{SW}, \texttt{NE} and \texttt{SE}.

```
\clist_new:N \l_@@_corners_clist
```

```
\dim_new:N \l_@@_notes_above_space_dim
\hook_gput_code:nnn { begindocument } { . }
{ \dim_set:Nn \l_@@_notes_above_space_dim { 1 \text{ mm} } }
```
We use a hook only by security in case \texttt{revtex4-1} is used (even though it is obsolete).

The flag $\l_@@_nullify_dots_bool$ corresponds to the option \texttt{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 \texttt{matrix} and \texttt{ldots}, $\vdots$ etc.).

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

The following flag corresponds to the key \texttt{respect-arraystretch} (that key has an effect on the blocks).

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

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).

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

The following boolean corresponds to the key \texttt{create-cell-nodes} of the keyword \texttt{CodeBefore}.

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

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.

\begin{verbatim}
\str_new:N \l_@@_name_str
\end{verbatim}

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”.

\begin{verbatim}
\bool_new:N \l_@@_medium_nodes_bool
\bool_new:N \l_@@_large_nodes_bool
\end{verbatim}

The boolean $\l_@@_exceptBorders_bool$ will be raised when the key \texttt{hvlines-except-borders} will be used (but that key has also other effects).

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

The dimension $\l_@@_left_margin_dim$ correspond to the option \texttt{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 ($\hdottedline$).

\begin{verbatim}
\dim_new:N \l_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim
\end{verbatim}

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

\begin{verbatim}
\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim
\end{verbatim}

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

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

The following parameter is for the color the dotted lines drawn by \texttt{Cdots}, \texttt{Ldots}, \texttt{Vdots}, \texttt{Ddots}, \texttt{Iddots} and \texttt{Hdotsfor} but \textit{not} the dotted lines drawn by $\hdottedline$ and “:”.

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

The following token list corresponds to the key \texttt{delimiters/color}.

\begin{verbatim}
\tl_new:N \l_@@_delimiters_color_tl
\end{verbatim}
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 \texttt{\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.

\begin{verbatim}
\bool_new:N \l_@@_delimiters_max_width_bool
\keys_define:nn { NiceMatrix / xdots }
{ line-style .code:n =
  \bool_lazy_or:nnTF
  \cs_if_exist_p:N \tikzpicture
  \str_if_eq_p:nn { #1 } { standard }
  { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
  { \@@_error:n { bad-option-for-line-style } }
, line-style .value_required:n = true ,
  color .tl_set:N = \l_@@_xdots_color_tl ,
  color .value_required:n = true ,
  shorten .code:n =
  \hook_gput_code:nnn { begindocument } { . }
  \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
  \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
, shorten-start .code:n =
  \hook_gput_code:nnn { begindocument } { . }
  \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } ,
  shorten-end .code:n =
  \hook_gput_code:nnn { begindocument } { . }
  \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } ,
  \dim_set:Nn \l_@@_xdots_radius_dim { #1 } ,
  \dim_set:Nn \l_@@_xdots_inter_dim { #1 } ,
\end{verbatim}

We use a hook only by security in case \texttt{revtex4-1} is used (even though it is obsolete). Idem for the following keys.

\begin{verbatim}
shorten .value_required:n = true ,
shorten-start .value_required:n = true ,
shorten-end .value_required:n = true ,
radius .code:n =
  \hook_gput_code:nnn { begindocument } { . }
  \dim_set:Nn \l_@@_xdots_radius_dim { #1 } ,
\end{verbatim}

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

\begin{verbatim}
down .tl_set:N = \l_@@_xdots_down_tl ,
up .tl_set:N = \l_@@_xdots_up_tl ,
\end{verbatim}

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

\begin{verbatim}
draw-first .code:n = \prg_do_nothing: ,
\end{verbatim}
First, we define a set of keys “NiceMatrix / Global” which will be used (with the mechanism of .inherit:n) by other sets of keys.

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

\keys_define:nn { NiceMatrix / Global }
{
  custom-line .code:n = \@@_custom_line:n { #1 } ,
  delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
  delimiters .value_required:n = true ,
  rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
  rules .value_required:n = true ,
  standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
  standard-cline .default:n = true ,
  cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
  cell-space-top-limit .value_required:n = true ,
  cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
  cell-space-bottom-limit .value_required:n = true ,
  cell-space-limits .meta:n =
    {
      cell-space-top-limit = #1 ,
      cell-space-bottom-limit = #1 ,
    } ,
  cell-space-limits .value_required:n = true ,
  xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
  light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
  light-syntax .default:n = true ,
  end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
  end-of-row .value_required:n = true ,
  first-col .code:n = \int_zero:N \l_@@_first_col_int ,
  first-row .code:n = \int_zero:N \l_@@_first_row_int ,
  last-row .int_set:N = \l_@@_last_row_int ,
  last-row .default:n = -1 ,
  code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
  code-for-first-col .value_required:n = true ,
  code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
  code-for-last-col .value_required:n = true ,
  code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
  code-for-first-row .value_required:n = true ,
  code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
  code-for-last-row .value_required:n = true ,
  hlines .clist_set:N = \l_@@_hlines_clist ,
  vlines .clist_set:N = \l_@@_vlines_clist ,
  hlines .default:n = all ,
  vlines .default:n = all ,
  vlines-in-sub-matrix .code:n =
    { 
      \tl_if_single_token:nTF { #1 } 
        { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } } 
        { \@@_error:n { One-letter-allowed } } 
    } ,
  vlines-in-sub-matrix .value_required:n = true ,
  hvlines .code:n =
    { 
      \clist_set:Nn \l_@@_hlines_clist { all } 
      \clist_set:Nn \l_@@_vlines_clist { all } 
    } ,
  hvlines-except-borders .code:n =
We define a set of keys used by the environments of nicematrix (but not by the command \NiceMatrixOptions).
\keys_define:nn { NiceMatrix / Env }
{ }

The key hvlines-except-corners is now deprecated (use hvlines and corners instead).
hvlines-except-corners .code:n = \@@_fatal:n { hvlines-except-corners } ,
hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
corners .clist_set:N = \l_@@_corners_clist ,
corners .default:n = { NW , SW , NE , SE } ,
\code_before .code:n =
{ \tl_if_empty:nF { #1 } }
{ \tl_put_right:Nn \l_@@_code_before_tl { #1 } }
\bool_set_true:N \l_@@_code_before_bool
}

The options c, t and b of the environment \{NiceArray\} have the same meaning as the option of the classical environment \{array\}.
c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
baseline .tl_set:N = \l_@@_baseline_tl ,
baseline .value_required:n = true ,
columns-width .code:n =
\tl_if_eq:nnTF { #1 } { auto }
{ \bool_set_true:N \l_@@_auto_columns_width_bool }
{ \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
columns-width .value_required:n = true ,
name .code:n =
We test whether we are in the measuring phase of an environment of \texttt{amsmath} (always loaded by \texttt{nicematrix}) because we want to avoid a fallacious message of duplicate name in this case.

\begin{verbatim}
\legacy_if:nF { measuring@ }
{
  \str_set:Nn \l_tmpa_str { #1 }
  \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
  { \@@_error:nn { Duplicate name } { #1 } }
  \str_set_eq:NN \l_@@_name_str \l_tmpa_str
  ,
  name .value_required:n = true ,
  code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
  code-after .value_required:n = true ,
  colortbl-like .code:n =
  \bool_set_true:N \l_@@_colortbl_like_bool
  \bool_set_true:N \l_@@_code_before_bool ,
  colortbl-like .value_forbidden:n = true
}
\keys_define:nn { NiceMatrix / notes }
{
  para .bool_set:N = \l_@@_notes_para_bool ,
  para .default:n = true ,
  code-before .tl_set:N = \l_@@_notes_code_before_tl ,
  code-before .value_required:n = true ,
  code-after .tl_set:N = \l_@@_notes_code_after_tl ,
  code-after .value_required:n = true ,
  bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
  bottomrule .default:n = true ,
  style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
  style .value_required:n = true ,
  label-in-tabular .code:n =
  \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
  label-in-tabular .value_required:n = true ,
  label-in-list .code:n =
  \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
  label-in-list .value_required:n = true ,
  enumitem-keys .code:n =
  \hook_gput_code:nnn { begindocument } { . }
  \bool_if:NT \c_@@_enumitem_loaded_bool
  \setlist* [ tabularnotes ] { #1 } }
\}
enumitem-keys .value_required:n = true ,
enumitem-keys-para .code:n =
{
  \hook_gput_code:nnn { begindocument } { . }
  \bool_if:NT \c_@@_enumitem_loaded_bool
  \setlist* [ tabularnotes* ] { #1 } }
\}
enumitem-keys-para .value_required:n = true ,
detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
detect-duplicates .default:n = true ,
unknown .code:n = \@@_error:n { Unknown key for notes }
\}
\keys_define:nn { NiceMatrix / delimiters }
{
  max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
  max-width .default:n = true ,
  color .tl_set:N = \l_@@_delimiters_color_tl ,
\}
\end{verbatim}
\begin{verbatim}
\keys_define:nn { NiceMatrix }
{
    NiceMatrixOptions .inherit:n =
    { NiceMatrix / Global },
    NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
    NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
    NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
    SubMatrix / rules .inherit:n = NiceMatrix / rules ,
    CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix .inherit:n =
    { NiceMatrix / Global ,
      NiceMatrix / Env ,
    },
    NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
    NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceTabular .inherit:n =
    { NiceMatrix / Global ,
      NiceMatrix / Env ,
    },
    NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
    NiceTabular / rules .inherit:n = NiceMatrix / rules ,
    NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
    NiceArray .inherit:n =
    { NiceMatrix / Global ,
      NiceMatrix / Env ,
    },
    NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    NiceArray / rules .inherit:n = NiceMatrix / rules ,
    NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
PNiceArray .inherit:n =
    { NiceMatrix / Global ,
      NiceMatrix / Env ,
    },
    pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    pNiceArray / rules .inherit:n = NiceMatrix / rules ,
    pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
}
\keys_define:nn { NiceMatrix / NiceMatrixOptions }
{
    width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
    width .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 ,
\end{verbatim}

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.
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 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}
\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 ,
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}

\begin{verbatim}
NiceMatrixOptions is the command of the nicematrix package to fix options at the document level. The scope of these specifications is the current TeX group.
\NewDocumentCommand \NiceMatrixOptions { m } {
\keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
\end{verbatim}

We finalise the definition of the set of keys “{NiceMatrix} / {NiceMatrix}” with the options specific to `{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 ,
unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
}
\end{verbatim}

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

\begin{verbatim}
\keys_define:nn { NiceMatrix / NiceArray } {
}
\end{verbatim}
In the environments \{NiceArray\} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.

```latex
\keys_define:nn { NiceMatrix / pNiceArray }
{
  first-col .code:n = \int_zero:N \l_@@_first_col_int ,
  last-col .code:n = \tl_if_empty:nF {#1} { \@@_error:n { last-col~non~empty~for~NiceArray } }
  \int_zero:N \l_@@_last_col_int ,
  first-row .code:n = \int_zero:N \l_@@_first_row_int ,
  small .bool_set:N = \l_@@_small_bool ,
  small .value_forbidden:n = true ,
  \bool_set_true:N \l_@@_width_used_bool ,
  width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
  \bool_set_true:N \l_@@_width_used_bool ,
  width .value_required:n = true ,
  notes / para .bool_set:N = \l_@@_notes_para_bool ,
  notes / para .default:n = true ,
  notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
  notes / bottomrule .default:n = true ,
  tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
  tabularnote .value_required:n = true ,
  \r .code:n = \@@_error:n { r-or-l-with-preamble } ,
  \l .code:n = \@@_error:n { r-or-l-with-preamble } ,
  unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
}
```

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

```latex
\keys_define:nn { NiceMatrix / NiceTabular }
{
  \r .code:n = \@@_error:n { r-or-l-with-preamble } ,
  \l .code:n = \@@_error:n { r-or-l-with-preamble } ,
  unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
}
```

The dimension \texttt{width} will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```latex
\keys_define:nn { NiceMatrix / NiceTabular }
{
  \r .code:n = \@@_error:n { r-or-l-with-preamble } ,
  \l .code:n = \@@_error:n { r-or-l-with-preamble } ,
  unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
}
```

Important code used by \{NiceArrayWithDelims\}

The pseudo-environment \texttt{\@@_cell_begin:w – \@@_cell_end:} 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 \texttt{\halign} (via an environment \texttt{\{array\}}).
The token list \g_@@_post_action_cell_tl will be set during the composition of the box \l_@@_cell_box and will be used after the composition in order to modify that box (that’s why it’s called a post-action).
\tl_gclear:N \g_@@_post_action_cell_tl

At the beginning of the cell, we link \CodeAfter to a command which do begin with \ (whereas the standard version of \CodeAfter does not).
\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

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

Now, we increment the counter of the rows. We don’t 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.
\int_compare:nNnT \c@jCol = 1
\begin{Verbatim}
\int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
\end{Verbatim}

The content of the cell is composed in the box \l_@@_cell_box. The \hbox_set_end: corresponding to this \hbox_set:Nw \l_@@_cell_box will be in the \@@_cell_end: (and the potential \c_math_toggle_token also).
\hbox_set:Nw \l_@@_cell_box \bool_if:NF \l_@@_NiceTabular_bool
\begin{Verbatim}
\bool_if:NT \l_@@_small_bool \scriptstyle
\end{Verbatim}
\c_math_toggle_token
\bool_if:NT \l_@@_small_bool \scriptstyle

For unexplained reason, with XeTeX (and not with the other engines), the environments of \nicematrix were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we use it now (in each cell of the array).
\color {nicematrix}
\g_@@_row_style_tl

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 al don’t apply in the corners of the matrix.
\int_compare:nNnTF \c@iRow = 0
\begin{Verbatim}
\int_compare:nNnT \c@jCol > 0
\begin{Verbatim}
\l_@@_code_for_first_row_tl \xglobal \colorlet {nicematrix-first-row} \. \}
\end{Verbatim}
\end{Verbatim}
\begin{Verbatim}
\int_compare:nNnT \c@iRow = \l_@@_last_row_int
\begin{Verbatim}
\l_@@_code_for_last_row_tl \xglobal \colorlet {nicematrix-last-row} \. \}
\end{Verbatim}
\end{Verbatim}

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.
Remark: If the key recreate-cell-nodes of the CodeBefore is used, then we will add some lines to that command.

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.
\dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
{
  \box_set_dp:Nn \l_@@_cell_box
  \dim_max:nn \{ \box_dp:N \l_@@_cell_box \} \g_@@_blocks_dp_dim
  \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_max:nn \{ \box_ht:N \l_@@_cell_box \} \g_@@_blocks_ht_dim
  \dim_gzero:N \g_@@_blocks_ht_dim
}
\cs_new_protected:Npn \@@_cell_end:
{
  \@@_math_toggle_token:
  \hbox_set_end:
  \g_@@_post_action_cell_tl
  \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
  \@@_adjust_size_box:
  \box_set_ht:Nn \l_@@_cell_box
  \dim_max:nn \{ \box_ht:N \l_@@_cell_box \} \l_@@_cell_space_top_limit_dim
  \box_set_dp:Nn \l_@@_cell_box
  \dim_max:nn \{ \box_dp:N \l_@@_cell_box \} \l_@@_cell_space_bottom_limit_dim

We want to compute in \g_@@_max_cell_width_dim the width of the widest cell of the array (except the cells of the “first column” and the “last column”).
\dim_gset:Nn \g_@@_max_cell_width_dim
\dim_max:nn \g_@@_max_cell_width_dim \{ \box_wd:N \l_@@_cell_box \}

The following computations are for the “first row” and the “last row”.
\@@_update_for_first_and_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. Up to now we use the following technique:
- if the width of the box \l_@@_cell_box (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a \rlap, a \llap or a \mathclap of mathtools.
- the cells with a command \ldots or \Cdots, \vdots, etc., should also be considered as empty: if nullify-dots is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of \CodeAfter); however, if nullify-dots is not in force, a phantom of \ldots, \cdots, \vdots is inserted and its width is not equal to zero; that’s why these commands raise a boolean \g_@@_empty_cell_bool and we begin by testing this boolean.
\bool_if:NT \g_@@_empty_cell_bool
{ \bool_lazy_or:nnTF \g_@@_not_empty_cell_bool
  \dim_compare_p:nNn \{ \box_wd:N \l_@@_cell_box \} > \c_zero_dim
  \@@_node_for_cell:
  \box_use_drop:N \l_@@_cell_box
}
The following command creates the PGF name of the node with, of course, \texttt{l@@cell\_box} as the content.

As its name says, the following command is a patch for the command \texttt{@@node\_for\_cell:}. This patch will be appended on the left of \texttt{@@node\_for\_the\_cell:} when the construction of the cell nodes (of the form (i-j)) in the \texttt{CodeBefore} is required.

I don’t know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way \texttt{latex}, \texttt{dvips}, \texttt{ps2pdf} (or Adobe Distiller). However, it seems to work.
We have no explanation for the different behavior between the TeX engines...

\bool_lazy_or:nTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:

\@@_patch_node_for_cell:n
\{ \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } \}
\{ \@@_patch_node_for_cell:n { } \}

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 & \Cdots & & 6 \\
7 & \Cdots[color=red]
\end{pNiceMatrix}

the content of \g_@@_Cdots_lines_tl will be:
\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{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.

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

It colortbl is loaded, \@tabarray has been redefined to incorporate \CT@start.

\@tabarray
\l_@@_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.

\[ \str_if_eq:VnTF \l_@@_baseline_tl c c t \]

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.
\cs_set_eq:NN \@@_old_ialign: \ialign

89
The following command creates a row node (and not a row of nodes!).
\cs_new_protected:Npn \@@_create_row_node: 
\{ 
\int_compare:nNnT \c@iRow > \g_@@_last_row_node_int 
\{ 
\int_gset_eq:NN \g_@@_last_row_node_int \c@iRow 
\@@_create_row_node_i: 
\} 
\}
\cs_new_protected:Npn \@@_create_row_node_i: 
\{ 

The \texttt{\hbox:n} (or \texttt{\hbox}) is mandatory.
\begin{verbatim}
\hbox 
\{ 
\bool_if:NT \l_@@_code_before_bool 
\{ 
\vtop 
\{ \skip_vertical:N 0.5\arrayrulewidth 
\pgfsys@markposition 
\{ \@@_env: row \int_eval:n { \c@iRow + 1 } \} 
\skip_vertical:N -0.5\arrayrulewidth 
\}
\pgfpicture 
\pgfrememberpicturepositiononpagetrue 
\pgfcoordinate { \@@_env: row \int_eval:n { \c@iRow + 1 } } 
\pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } 
\str_if_empty:NF \l_@@_name_str 
\{ 
\pgfnodealias 
{ \l_@@_name_str - row \int_eval:n { \c@iRow + 1 } } 
\[ \@@_env: row \int_eval:n { \c@iRow + 1 } ] 
\} 
\endpgfpicture 
\}
\end{verbatim}

The following must \textit{not} be protected because it begins with \texttt{\noalign}.
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i: 
\{ 
\int_gzero:N \c@jCol 
\bool_gset_false:N \g_@@_after_col_zero_bool 
\bool_if:NF \g_@@_row_of_col_done_bool 
\{ 
\@@_create_row_node: 
\}
\}

We don’t draw now the rules of the key \texttt{hlines} (or \texttt{hvlines}) but we reserve the vertical space for theses rules (the rules will be drawn by PGF).
\begin{verbatim}
\tl_if_empty:NF \l_@@_hlines_clist 
\{ 
\tl_if_eq:NnF \l_@@_hlines_clist { all } 
\{ 
\exp_args:Nx 
\clist_if_in:NnT \l_@@_hlines_clist 
\\c@iRow + 1 ] 
\} 
\} 
\}
\end{verbatim}
The counter \texttt{\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 \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 use a \TeX{} group in order to limit the scope of \texttt{\CT@arc@}.

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

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

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

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

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

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

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

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

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_array_ii:
  { \xglobal \colorlet { nicematrix } { . } 
    \int_gzero:N \g_@@_total_X_weight_int \\
    \@@_expand_clist:N \l_@@_hlines_clist \\
    \@@_expand_clist:N \l_@@_vlines_clist 
  }
\end{verbatim}

For unexplained reason, with Xe\TeX{} (and not with the other engines), the environments of \texttt{nicematrix} were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we will it in each cell.

\begin{verbatim}
\xglobal \colorlet { nicematrix } { . }
\end{verbatim}

The number of letters \texttt{X} in the preamble of the array.

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

\begin{Verbatim}
\bool_if:NT \c_@@_booktabs_loaded_bool
\{ \tl_put_left:Nn \BTnormal \@@_create_row_node_i: \} % modified in 6.10a
\box_clear_new:N \l_@@_cell_box
\normalbaselines
\end{Verbatim}

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\}`).

\begin{Verbatim}
\bool_if:NT \l_@@_small_bool
\{
\cs_set_nopar:Npn \arraystretch { 0.47 }
\dim_set:Nn \arraycolsep { 1.45 pt }
\}
\end{Verbatim}

\begin{Verbatim}
\bool_if:NT \g_@@_recreate_cell_nodes_bool
\{
\tl_put_right:Nn \@@_begin_of_row:
\{ \pgfsys@markposition
\{ \@@_env: - row - \int_use:N \c@iRow - base \}
\}
\end{Verbatim}

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`.

\begin{Verbatim}
\cs_set_nopar:Npn \ialign
\{ \bool_if:NTF \l_@@_colortbl_loaded_bool
\{ \CT@everycr
\{ \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: } \CT@everycr:
\}
\{ \everycr \{ \@@_everycr: \}
\tabskip = \c_zero_skip
\end{Verbatim}

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`.

\begin{Verbatim}
\dim_gzero_new:N \g_@@_dp_row_zero_dim
\dim_gzero_new:N \g_@@_ht_row_zero_dim
\end{Verbatim}

\footnote{The option small of nicematrix changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `\{array\}`.}
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.

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).

We redefine \multicolumn and, since we want \multicolumn to be available in the potential environments {tabular} nested in the environments of nicematrix, we patch {tabular} to go back to the original definition.

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

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

\int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }
At the end of the environment \{array\}, \c@iRow will be the total number of rows. \g_@@_row_total_int will be the number of rows excepted the last row (if \l_@@_last_row_bool has been raised with the option last-row).

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

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

If there is a exterior row, we patch a command used in \@@_cell_begin:w in order to keep track of some dimensions needed to the construction of that “last row”.

This is the end of \@@_pre_array_ii.

The command \@@_pre_array: will be executed after analyse of the keys of the environment.

We recall that \l_@@_last_row_int and \l_@@_last_column_int are not the numbers of the last row and last column of the array. There are only the values of the keys last-row and last-column (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of nicematrix. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the aux file (of course, it’s possible only after the first compilation).

If there is a exterior row, we patch a command used in \@@_cell_begin:w in order to keep track of some dimensions needed to the construction of that “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 \} }
\seq_gclear:N \g_@@_cols_vlism_seq
\seq_gclear:N \g_@@_submatrix_seq
Now the \CodeBefore.
\bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:
The value of \g_@@_pos_of_blocks_seq has been written on the aux file and loaded before the (potential) execution of the \CodeBefore. Now, we clear that variable because it will be reconstructed during the creation of the array.
\seq_gclear:N \g_@@_pos_of_blocks_seq
Idem for other sequences written on the aux file.
\seq_gclear_new:N \g_@@_multicolumn_cells_seq
\seq_gclear_new:N \g_@@_multicolumn_sizes_seq
The command \create_row_node: will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command \@@_create_row_node: will use the following counter to avoid such construction.
\int_gset:Nn \g_@@_last_row_node_int { -1 }
The code in \@@_pre_array_ii: is used only here.
\@@_pre_array_ii:
The array will be composed in a box (named \l_@@_the_array_box) because we have to do manipulations concerning the potential exterior rows.
\box_clear_new:N \l_@@_the_array_box
The preamble will be constructed in \g_@@_preamble_tl.
\@@_construct_preamble:
Now, the preamble is constructed in \g_@@_preamble_tl
We compute the width of both delimiters. We remember that, when the environment \{NiceArray\} is used, it’s possible to specify the delimiters in the preamble (eg \{ccc\}).
\dim_zero_new:N \l_@@_left_delim_dim
\dim_zero_new:N \l_@@_right_delim_dim
\bool_if:NTF \l_@@_NiceArray_bool
{ \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep } \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep } }
{ }
The command \bBigg@ is a command of amsmath.
\hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
\dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
\hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
\dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
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).

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

The following command `\@@_CodeBefore_Body:w` will be used when the keyword `\CodeBefore` is present at the beginning of the environment.

\begin{verbatim}
\cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
{ \tl_put_right:Nn \l_@@_code_before_tl { #1 }
  \bool_set_true:N \l_@@_code_before_bool
  \@@_CodeBefore:}
\end{verbatim}

We go on with `\@@_pre_array:`, which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

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

\section*{The `\CodeBefore`}

The following command will be executed if the `\CodeBefore` has to be actually executed.

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_code_before:
{ \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
  \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
  \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
  \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
\end{verbatim}

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 3.1.4d.

\begin{verbatim}
\pgfsys@markposition { \@@_env: - position }
\pgfsys@getposition { \@@_env: - position } \@@_picture_position:
\pgfpicture
\pgfrelevantforpicturesizefalse
\end{verbatim}

First, the recreation of the `\row` nodes.

\begin{verbatim}
\int_step_inline:nnnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
{ \pgfsys@getposition { \@@_env: - row - #1 } \@@_node_position:
  \pgfcoordinate { \@@_env: - row - #1 }
  \pgfpointdiff \@@_picture_position: \@@_node_position: }
\end{verbatim}
Now, the recreation of the col nodes.

\int_step_inline:nnn \l_@@_first_col_int \{ \g_@@_col_total_int + 1 \} 
\{ 
\pgfsys@getposition \{ \@@_env: - col - ##1 \} \@@_node_position: 
\pgfcoordinate \{ \@@_env: - col - ##1 \} 
\{ \pgfpointdiff \@@_picture_position: \@@_node_position: \} 
\}

Now, you recreate the diagonal nodes by using the row nodes and the col nodes.

\@@_create_diag_nodes:

Now, the creation of the cell nodes (i-j), and, maybe also the “medium nodes” and the “large nodes”.

\bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
\endpgfpicture

Now, the recreation of the nodes of the blocks which have a name.

\@@_create_blocks_nodes:
\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 \rowlistcolors \@@_rowlistcolors
\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
\cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
\}
\cs_new_protected:Npn \@@_exec_code_before:
\{ 
\seq_gclear_new:N \g_@@_colors_seq
\bool_gset_false:N \g_@@_recreate_cell_nodes_bool
\group_begin:

We compose the \CodeBefore in math mode in order to nullify the spaces put by the user between instructions in the \CodeBefore.

\bool_if:NT \l_@@_NiceTabular_bool \l_@@_code_before_tl
\c_math_toggle_token
Here is the \CodeBefore. The construction is a bit complicated because \l_@@_code_before_tl may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do not execute the rest of \l_@@_code_before_tl (when it is asked for the creation of cell nodes in the \CodeBefore). That’s why we begin with a \q_stop: it will be used to discard the rest of \l_@@_code_before_tl.
\exp_last_unbraced:NV \@@_CodeBefore_keys: \l_@@_code_before_tl \q_stop

Now, all the cells which are specified to be colored by instructions in the \CodeBefore will actually be colored. It’s a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

\@@_actually_color:
\bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
\group_end:
\bool_if:NT \g_@@_recreate_cell_nodes_bool
  { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }

\keys_define:nn { NiceMatrix / CodeBefore }
  { create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
    create-cell-nodes .default:n = true ,
    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 ,
    unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter } }

\NewDocumentCommand \@@_CodeBefore_keys: { O { } }
  { \keys_set:nn { NiceMatrix / CodeBefore } { #1 } \@@_CodeBefore:w }

We have extracted the options of the keyword \CodeBefore in order to see whether the key create-cell-nodes has been used. Now, you can execute the rest of the \CodeAfter, excepted, of course, if we are in the first compilation.

\cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
  { \bool_if:NT \g_@@_aux_found_bool
      { \@@_pre_code_before:
          \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
            { \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
                \pgffcoordinate { \@@_env: - row - ##1 - base }
                  { \pgfpointdiff \@@_picture_position: \@@_node_position: }
                    \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
                      { \cs_if_exist:cT
                          \pgfpointdiff \@@_picture_position: \@@_node_position: 
                          \pgfpointdiff \@@_picture_position: \@@_node_position_i: 
                          \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
                      } }
      }

By default, if the user uses the \CodeBefore, only the col nodes, row nodes and diag nodes are available in that \CodeBefore. With the key create-cell-nodes, the cell nodes, that is to say the nodes of the form (i-j) (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

\cs_new_protected:Npn \@@_recreate_cell_nodes:
  { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
      { \pgf@sys@getposition { \@@_env: - ##1 - base } \@@_node_position:
          \pgfcoordinate { \@@_env: - row - ##1 - base }
            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
              \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
                { \cs_if_exist:cT
                    \pgf@sys@getposition
                      \pgf@sys@getposition
                        \pgf@sys@getposition
                          \pgf@sys@getposition
                            \pgf@sys@getposition
                              \pgf@sys@getposition
                                \pgf@sys@getposition
                                  { \@@_env: - #1 - ####1 - NW }
        } }
    }

98
The following command is called \@create_one_block_node:nnnnn but, in fact, it creates a node only if the last argument (\#5) which is the name of the block, is not empty.\footnote{Moreover, there is also in the list \g_@@_pos_of_blocks_seq the positions of the dotted lines (created by \Cdots, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).}
The environment \texttt{NiceArrayWithDelims}

\begin{Verbatim}
\NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } t } \CodeBefore
{ \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
  \@@_provide_pgfsyspdfmark:
  \bool_if:NT \c_@@_footnote_bool \savenotes
\end{Verbatim}

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

\begin{Verbatim}
\begin{bgroup}
\tl_gset:Nn \g_@@_left_delim_tl { #1 }
\tl_gset:Nn \g_@@_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_eq:NN \@@_old_CT@arc@ \CT@arc@
\str_if_empty:NT \g_@@_name_env_str
{ \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
\bool_if:NTF \l_@@_NiceTabular_bool
{ \mode_leave_vertical:
  \@@_test_if_math_mode:
  \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
  \bool_set_true:N \l_@@_in_env_bool
\end{Verbatim}

The command \texttt{\CT@arc@} contains the instruction of color for the rules of the array\textsuperscript{65}. 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 not loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by \texttt{colortbl}. Of course, we restore the value of \texttt{\CT@arc@} at the end of our environment.

\begin{Verbatim}
\cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@
\cs_if_exist:NT \tikz@library@external@loaded
{ \tikzexternaldisable
  \cs_if_exist:NT \ifstandalone
  { \tikzset { external / optimize = false } }
\end{Verbatim}

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

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

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

\begin{Verbatim}
\int_gincr:N \g_@@_env_int
\bool_if:HF \l_@@_block_auto_columns_width_bool
{ \dim_gzero_new:N \g_@@_max_cell_width_dim }
\end{Verbatim}

\textsuperscript{65}e.g. \texttt{\color[rgb]{0.5,0.5,0}}
The sequence \texttt{\g@@_blocks_seq} will contain the characteristics of the blocks (specified by \texttt{\Block}) of the array. The sequence \texttt{\g@@_pos_of_blocks_seq} will contain only the position of the blocks (except the blocks with the key \texttt{hvlines}).

\begin{verbatim}
\seq_gclear:N \g@@_blocks_seq
\seq_gclear:N \g@@_pos_of_blocks_seq
\end{verbatim}

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

\begin{verbatim}
\seq_gclear:N \g@@_pos_of_stroken_blocks_seq
\seq_gclear:N \g@@_pos_of_xdots_seq
\tl_gclear_new:N \g@@_code_before_tl
\tl_gclear:N \g@@_row_style_tl
\end{verbatim}

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

\begin{verbatim}
\bool_gset_false:N \g@@_aux_found_bool
\tl_if_exist:cT { c@@_\int_use:N \g@@_env_int_tl }
{ \bool_gset_true:N \g@@_aux_found_bool
\use:c { c@@_\int_use:N \g@@_env_int_tl }
}
\end{verbatim}

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

\begin{verbatim}
\tl_gclear:N \g@@_aux_tl
\tl_if_empty:NF \g@@_code_before_tl
{ \bool_set_true:N \l@@_code_before_bool
\tl_put_right:NV \l@@_code_before_tl \g@@_code_before_tl
}
\end{verbatim}

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

\begin{verbatim}
\bool_if:NTF \l@@_NiceArray_bool
{ \keys_set:nn { NiceMatrix / NiceArray } }
{ \keys_set:nn { NiceMatrix / pNiceArray } }
\tl_if_empty:NF \l@@_rules_color_tl
{ \exp_after:wN \@@_set_CT@arc@: \l@@_rules_color_tl \q_stop }
\end{verbatim}

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

\begin{verbatim}
\IfBooleanTF { #6 } \@@_CodeBefore\_Body:w \@@_pre\_array:
{ }
\end{verbatim}

End of the construction of the array (in the box \texttt{\l@@\_the\_array\_box}).

If the user has used the key \texttt{width} without any column \texttt{X}, we raise an error.

\begin{verbatim}
\bool_if:NT \l@@_width_used_bool
\end{verbatim}
Now, if there is at least one $X$-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, $l_{00}X_columns_dim$ will be the width of a column of weight 1. For a $X$-column of weight $n$, the width will be $l_{00}X_columns_dim$ multiplied by $n$. 

\begin{verbatim}
\int_compare:nNnT \g_@@_total_X_weight_int = 0
\{ \@@_error:n \{ width-without-X-columns \} \}
\end{verbatim}

\begin{verbatim}
\int_compare:nNnT \g_@@_total_X_weight_int > 0
\{ \tl_gput_right:Nx \g_@@_aux_tl
\{ \bool_set_true:N \l_@@_X_columns_aux_bool
 \dim_set:Nn \l_@@_X_columns_dim
 \{ \dim_compare:nNnTF
 \{ \l_@@_width_dim - \box_wd:N \l_@@_the_array_box \}
 \}< 0.001 pt \}
 \{ \dim_use:N \l_@@_X_columns_dim \}
 \{ \dim_eval:n
 \{ ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
 / \int_use:N \g_@@_total_X_weight_int
 + \l_@@_X_columns_dim \}
 \}
\}
\end{verbatim}

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

Now, the definition of $c_\text{Col}$ and $g_00\_col\_total\_int$ change: $c_\text{Col}$ will be the number of columns without the “last column”; $g_00\_col\_total\_int$ will be the number of columns with this “last column”.$^{66}$

\begin{verbatim}
\int_gset_eq:NN \c_\text{Col} \g_00\_col\_total\_int
\bool_if:nTF \g_00\_last\_col\_found_bool
{ \int_gdecr:N \c_\text{Col} }
\{ \int_compare:nNnT \l_@@_last_col_int > \{-1\}
{ \@@_error:n \{ last-col-not-used \} }
\}
\end{verbatim}

We fix also the value of $c_\text{iRow}$ and $g_00\_row\_total\_int$ with the same principle.

\footnote{We remind that the potential “first column” (exterior) has the number 0.}
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. 130).

\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.

\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 $\l_tmpa_dim$ which is the total height of the “first row” above the array (when the key first-row is used).

\begin{verbatim}
\int_compare:nNnTF \l_@@_first_row_int = 0
{ \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim 
  \dim_zero:N \l_tmpa_dim }
\end{verbatim}

We compute $\l_tmpb_dim$ which is the total height of the “last row” below the array (when the key last-row is used). A value of $-2$ for $\l_@@_last_row_int$ means that there is no “last row.”

\begin{verbatim}
\int_compare:nNnTF \l_@@_last_row_int > { -2 }
{ \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim 
  \dim_zero:N \l_tmpb_dim }
\end{verbatim}

\begin{verbatim}
\hbox_set:Nn \l_tmpa_box
{ \c_math_toggle_token \tl_if_empty:NF \l_@@_delimiters_color_tl 
  \{ \color { \l_@@_delimiters_color_tl } \}
  \exp_after:wN \left \g_@@_left_delim_tl \vcenter
\skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
\hbox
{ \bool_if:NTF \l_@@_NiceTabular_bool
  \{ \skip_horizontal:N -\tabcolsep \}
  \{ \skip_horizontal:N -\arraycolsep \}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

We take into account the “first row” (we have previously computed its total height in $\l_tmpa_dim$). The \hbox:n (or \hbox) is necessary here. There was a bug in the following line (corrected the 2021/11/23).

\begin{verbatim}
\skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
\hbox
{ \bool_if:NTF \l_@@_NiceTabular_bool
  \{ \skip_horizontal:N -\tabcolsep \}
  \{ \skip_horizontal:N -\arraycolsep \}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}

67A value of $-1$ for $\l_@@_last_row_int$ means that there is a “last row” but the user have not set the value with the option last row (and we are in the first compilation).
We take into account the “last row” (we have previously computed its total height in \l_tmpb_dim).
There was a bug in the following line (corrected the 2021/11/23).
\skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }

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.
\tl_if_empty:NF \l_@@_delimiters_color_tl
{ \color { \l_@@_delimiters_color_tl } }
\exp_after:wN \right \g_@@_right_delim_tl
\c_math_toggle_token

Now, the box \l_tmpa_box is created with the correct delimiters.
We will put the box in the TeX flow. However, we have a small work to do when the option delimiters/max-width is used.
\bool_if:NTF \l_@@_delimiters_max_width_bool
{ \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
\@@_put_box_in_flow:

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in \g_@@_width_last_col_dim: see p. 131).
\bool_if:NT \g_@@_last_col_found_bool
{ \skip_horizontal:N \g_@@_width_last_col_dim
 \skip_horizontal:N \col@sep }
\bool_if:NF \l_@@_Matrix_bool
{ \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
 { \@@_error:n { columns~not~used } } }
\group_begin:
\globaldefs = 1
\@@_msg_redirect_name:nn { columns~not~used } { error }
\group_end:
\@@_after_array:

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.
\egroup

We want to write on the aux file all the informations corresponding to the current environment.
\iow_now:Nn \@mainaux { \ExplSyntaxOn }
\iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
\iow_now:Nn \@mainaux
{ \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl } }
{ \exp_not:V \g_@@_aux_tl }
\iow_now:Nn \@mainaux { \ExplSyntaxOff }
\bool_if:NT \c_@@_footnote_bool \endsavenotes

This is the end of the environment \{NiceArrayWithDelims\}.
We construct the preamble of the array

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

The preamble given by the final user is in \texttt{\textbackslash g\_@@\_preamble\_tl} and the modified version will be stored in \texttt{\textbackslash 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{\textbackslash newcolumntype}). Since we use the tools of \texttt{array} to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

We redefine the column types \texttt{w} and \texttt{W}. We use \texttt{\textbackslash@@\_newcolumntype} instead of \texttt{\textbackslash newcolumntype} because we don’t want warnings for column types already defined. These redefinitions are in fact \textit{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 to use the standard column types \texttt{w} and \texttt{W} in potential \texttt{\{tabular\}} of \texttt{array} in some cells of our array. That’s why we do those redefinitions in a TeX group.

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).

If the package \texttt{varwidth} has defined the column type \texttt{V}, we protect from expansion by redefining it to \texttt{\textbackslash@@\_V} (which will be catched by our system).

First, we have to store our preamble in the token register \texttt{\@temptokena} (those “token registers” are \textit{not} supported by the L3 programming layer).

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}). The expanded version is still in \texttt{\textbackslash@temptokena}.

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker \texttt{\textbackslash q\_stop} and we will consume these tokens constructing the (new form of the) preamble in \texttt{\textbackslash g\_@@\_preamble\_tl}. This is done recursively with the command \texttt{\textbackslash@@\_patch\_preamble:n}. In the same time, we will count the columns with the counter \texttt{\textbackslash c\_\textbackslash j\_Col}.

\texttt{\textbackslash g\_tmpb\_bool} will be raised if you have a | at the end of the preamble.
The sequence \texttt{\_g@cols_vlsim_seq} will contain the numbers of the columns where you will have to draw vertical lines in the potential sub-matrices (hence the name vlsim).

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}.

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

If there was delimiters at the beginning or at the end of the preamble, the environment \texttt{\{NiceArray\}} is transformed into an environment \texttt{\{xNiceMatrix\}}.

We want to remind whether there is a specifier \texttt{|} at the end of the preamble.

We complete the preamble with the potential “exterior columns” (on both sides).
We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in \verb|NiceTabular*| (\verb|\l_@@_tabular_width_dim| = 0pt).

The command \verb|\@@_patch_preamble:n| is the main function for the transformation of the preamble. It is recursive.

When \verb|tabularx| is loaded, a local redefinition of the specifier \verb|X| is done to replace \verb|X| by \verb|\@@_X|. Thus, our column type \verb|X| will be used in the \verb|NiceTabularX|.

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key \verb|custom-line| in \verb|NiceMatrixOptions|. That key takes in as value a list of key=value pairs. Among the keys available in that list, there is the key letter. All the letters defined by this way by the final user for such customized rules are added in the set of keys \verb|NiceMatrix/ColumnTypes|. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won’t never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key \verb|custom-line| in \verb|NiceMatrixOptions|. That key takes in as value a list of key=value pairs. Among the keys available in that list, there is the key letter. All the letters defined by this way by the final user for such customized rules are added in the set of keys \verb|NiceMatrix/ColumnTypes|. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won’t never be used as keys by the final user (he will use, instead, letters in the preamble of its array).
Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

For $c$, $l$ and $r$

```latex
\cs_new_protected:Npn \@@_patch_preamble_i:n #1
\{ \tl_gput_right:Nn \g_@@_preamble_tl
  { > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } } #1
  < \@@_cell_end: }
\}
```

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

```latex
\int_gincr:N \c@jCol
\@@_patch_preamble_xi:n
\}
```

For $>$, $!$ and $@$

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

For $|

```latex
\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
\}
```

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

Here, the command $\dim_eval:n$ is mandatory.

```latex
\dim_eval:n
\{ \arrayrulewidth * \l_tmpa_int
  + \doublerulesep * ( \l_tmpa_int - 1)
\}
```

```latex
\tl_gput_right:Nx \g_@@_internal_code_after_tl
```

108
We don’t have provided value for start nor for end, which means that the rule will cover (potentially) all the rows of the array.

The specifier $p$ (and also the specifiers $m$ and $b$) have an optional argument between square brackets for a list of key-value pairs. Here are the corresponding keys. This set of keys will also be used by the $X$ columns.

For $p$, $b$ and $m$. The argument #1 is that value : $p$, $b$ or $m$.

Now, you look for a potential character [ after the letter of the specifier (for the options).
The possible values of \_hpos_col_str are \texttt{j} (for \texttt{justified} which is the initial value), \texttt{l}, \texttt{c} and \texttt{r} (when the user has used the corresponding key in the optional argument of the specifier).

The first argument is the width of the column. The second is the type of environment: \texttt{minipage} or \texttt{varwidth}.

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

#1 is the optional argument of \texttt{\{minipage\}} (or \texttt{\{varwidth\}}): \texttt{t} of \texttt{b}. Indeed, for the columns of type \texttt{m}, we use the value \texttt{b} here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the \texttt{\{minipage\}} (or \texttt{\{varwidth\}}), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (\texttt{centering}, \texttt{raggedright}, \texttt{raggedleft} or nothing). It’s also possible to put in that #3 some code to fix the value of \_hpos_cell_str which will be available in each cell of the column.

#4 is an extra-code which contains \texttt{\_center_cell_box}: (when the column is a \texttt{m} column) or nothing (in the other cases).
#5 is a code put just before the c (or r or l: see #8).
#6 is a code put just after the c (or r or l: see #8).
#7 is the type of environment: minipage or varwidth.
#8 is the letter c or r or l which is the basic specifier of column which is used in fine.

\cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn \#1 \#2 \#3 \#4 \#5 \#6 \#7 \#8
\tl_gput_right:Nn \g_@@_preamble_tl
\{ \}

The parameter \l_@@_col_width_dim, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

\dim_set:Nn \l_@@_col_width_dim { \#2 }
\@@_cell_begin:w
\begin { \#7 } \[ \#1 \] { \#2 }
\everypar
\vrule height \box_ht:N \@arstrutbox width \c_zero_dim
\everypar \}
\}

The following lines have been taken from array.sty.

\everypar
\{ \}
\vrule height \box_ht:N \@arstrutbox width \c_zero_dim
\everypar \}

Now, the potential code for the horizontal position of the content of the cell (\centering, \raggedright, \raggedleft or nothing).

\g_@@_row_style_tl \arraybackslash \#5 \#8 \#6

The following code is to allow something like \centering in \RowStyle.

\g_@@_row_style_tl \arraybackslash \#5
\}
\}
\}
\}

The following line has been taken from array.sty.

\@finalstrut \@arstrutbox
% \bool_if:NT \g_@@_rotate_bool \{ \raggedright \hspace = 3 cm \}
\end { \#7 }

If the letter in the preamble is m, \#4 will be equal to \@@_center_cell_box: (see just below).

\g_@@_row_style_tl
\arraybackslash
\#5
\}
\#8
\#6

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It’s not possible to know the number of rows of the cell. However, we consider (as in array) that if the height of the cell is no more that the height of \@arstrutbox, there is only one row.

\cs_new_protected:Npn \@@_center_cell_box:
\tl_gput_right:Nn \g_@@_post_action_cell_tl, we require a post-action of the box \l_@@_cell_box.

By putting instructions in \g_@@_post_action_cell_tl, we require a post-action of the box \l_@@_cell_box.
\tl_gput_right:Nn \g_@@_post_action_cell_tl
\int_compare:nNnT
\{ \box_ht:N \l_@@_cell_box \}
\}
\}
\}
\}

111
\hbox_set:Nn \l_@@_cell_box
{
\box_move_down:nn
{
( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
+ \baselineskip ) / 2
}
\box_use:N \l_@@_cell_box
}
}

For \texttt{V} (similar to the \texttt{V} of \texttt{varwidth}).
\cs_new_protected:Npn \@@_patch_preamble_v:n #1
{
\str_if_eq:nnTF { #1 } { \[ }
{ \@@_patch_preamble_v_i:w \[ }
{ \@@_patch_preamble_v_i:w \[ \] { #1 } }
}
\cs_new_protected:Npn \@@_patch_preamble_v_i:w \[ #1 \]
{ \@@_patch_preamble_v_ii:nn { #1 } }
\cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
{
\str_set:Nn \l_@@_vpos_col_str { p }
\str_set:Nn \l_@@_hpos_col_str { j }
\keys_set:nn { WithArrows / p-column } { #1 }
\bool_if:NTF \c_@@_varwidth_loaded_bool
{ \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
{ \@@_error:n { varwidth~not~loaded }
\@@_patch_preamble_iv_iv:nn { #2 } { minipage } }
}

For \texttt{w} and \texttt{W}
\cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
{
\tl_gput_right:Nn \g_@@_preamble_tl
{
> {
The parameter \texttt{\l_@@_col_width_dim}, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

\dim_set:Nn \l_@@_col_width_dim { #4 }
\hbox_set:Nw \l_@@_cell_box
\@@_cell_begin:w
\str_set:Nn \l_@@_hpos_cell_str { #3 }

\begin{Verbatim}
\end{Verbatim}

We increment the counter of columns and then we test for the presence of a <.
\int_gincr:N \c@jCol
\@@_patch_preamble_xi:n
}
For \@@_S: If the user has used S[, ...], S has been replaced by \@@_S: during the first expansion of the preamble (done with the tools of standard LaTeX and array).

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_vii:n #1
\{
  \str_if_eq:nnTF { #1 } { [ }
  \{ \@@_patch_preamble_vii:i:w \[ }
  \{ \@@_patch_preamble_vii:i:w \[ ] { #1 } \}
\}
\cs_new_protected:Npn \@@_patch_preamble_vii:i:w \[ #1 \]
\{ \@@_patch_preamble_vii:ii:n { #1 } \}
\cs_new_protected:Npn \@@_patch_preamble_vii:ii:n #1
\{
  \str_if_eq:VnTF \g_@@_left_delim_tl { . }
  \{ \tl_gset:Nn \g_@@_left_delim_tl { #1 } \tl_gset:Nn \g_@@_right_delim_tl { . } \@@_patch_preamble:n #1 \}
  \{ \tl_gput_right:Nx \g_@@_internal_code_after_tl { \@@_delimiter:nnn #1 \int_eval:n { \c@jCol + 1 } \c_true_bool } \}
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
\{
  \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
  \iftrue
  \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } \@@_patch_preamble_viii:i:nn { #1 } { #2 }
  \}
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:i:nn #1 #2
\{
  \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } \@@_patch_preamble_viii:nn { #1 } { #2 }
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
\{
  \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } \@@_patch_preamble_viii:nn { #1 } { #2 }
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
\{
  \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } \@@_patch_preamble_viii:nn { #1 } { #2 }
\}
\\end{verbatim}

We test whether the version of nicematrix is at least 3.0. We will change de programmation of the test further with something like \@ifpackagelater.

\begin{verbatim}
\cs_if_exist:NTF \siunitx_cell_begin:w
\{ \tl_gput_right:Nn \g_@@_preamble_tl \ siunitx_cell_begin:w
\{ \keys_set:nn { siunitx } { #1 } \siunitx_cell_begin:w
\}
\c
\{ \siunitx_cell_end: \@@_cell_end: \}
\}
\int_gincr:N \c@jCol \@@_patch_preamble_xi:n
\{ \@@_fatal:n { Version~of~siunitx~too~old } \}
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
\{
  \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
  \iftrue
  \tl_gset:Nn \g_@@_left_delim_tl { . } \tl_gset:Nn \g_@@_right_delim_tl { . } \@@_patch_preamble:n #1
  \}
  \{ \tl_gput_right:Nx \g_@@_internal_code_after_tl { \@@_delimiter:nnn #1 \int_eval:n { \c@jCol + 1 } \c_true_bool } \}
\}
\cs_new_protected:Npn \@@_patch_preamble_viii:i:nn #1 #2
\{
  \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
  \iftrue
  \tl_gput_right:Nn \g_@@_preamble_tl \ siunitx_cell_begin:w
  \{ \keys_set:nn { siunitx } { #1 } \siunitx_cell_begin:w
  \}
  \c
  \{ \siunitx_cell_end: \@@_cell_end: \}
\}
\int_gincr:N \c@jCol \@@_patch_preamble_xi:n
\{ \@@_fatal:n { Version~of~siunitx~too~old } \}
\}
\int_gincr:N \c@jCol \@@_patch_preamble_xi:n
\{ \@@_fatal:n { Version~of~siunitx~too~old } \}
\}
\int_gincr:N \c@jCol
\}
\\end{verbatim}

For (, [ and \{.

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
\{
  \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
  \iftrue
  \tl_gput_right:Nn \g_@@_preamble_tl \ siunitx_cell_begin:w
  \{ \keys_set:nn { siunitx } { #1 } \siunitx_cell_begin:w
  \}
  \c
  \{ \siunitx_cell_end: \@@_cell_end: \}
\}
\int_gincr:N \c@jCol \@@_patch_preamble_xi:n
\{ \@@_fatal:n { Version~of~siunitx~too~old } \}
\}
\int_gincr:N \c@jCol
\}
\int_gincr:N \c@jCol
\}
\\end{verbatim}

If we are before the column 1 and not in \{NiceArray\}, we reserve space for the left delimiter.

\begin{verbatim}
\int_compare:nNnTF \c@jCol = \c_zero_int
\{ \str_if_eq:VnTF \g_@@_left_delim_tl { . }
\{ \tl_gset:Nn \g_@@_left_delim_tl { #1 } \tl_gset:Nn \g_@@_right_delim_tl { . } \@@_patch_preamble:n #1
\}
\{ \tl_gput_right:Nx \g_@@_internal_code_after_tl { \@@_delimiter:nnn #1 \int_eval:n { \c@jCol + 1 } \c_true_bool } \}
\}
\\end{verbatim}

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

\begin{verbatim}
\tl_gset:Nn \g_@@_left_delim_tl { #1 } \tl_gset:Nn \g_@@_right_delim_tl { . }
\{ \@@_patch_preamble:n #2
\}
\{ \tl_gput_right:Nn \g_@@_preamble_tl \ siunitx_cell_begin:w
\{ \keys_set:nn { siunitx } { #1 } \siunitx_cell_begin:w
\}
\c
\{ \siunitx_cell_end: \@@_cell_end: \}
\}
\\end{verbatim}
For ), ] and }. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have a opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is \{NiceArray\}).
For the case of a letter \textit{X}. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a \{ after the letter \textit{X}.

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_x:n #1
\begin{Verbatim}
\str_if_eq:nnTF { #1 } { \[ }
\{ \@@_patch_preamble_x_i:w \[ }
\{ \@@_patch_preamble_x_i:w [ ] \#1 \}
\end{Verbatim}
\end{verbatim}

\#1 is the optional argument of the \textit{X} specifier (a list of \textit{key-value} pairs).

The following set of keys is for the specifier \textit{X} in the preamble of the array. Such specifier may have as keys all the keys of \{ \texttt{WithArrows} / \texttt{p-column} \} but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \texttt{\l_@@_weight_int}).

\begin{verbatim}
\keys_define:nn { WithArrows / X-column } { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }
\end{verbatim}

In the following command, \#1 is the list of the options of the specifier \textit{X}.

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
\begin{Verbatim}
The possible values of \texttt{\l_@@_hpos_col_str} are \texttt{j} (for \textit{justified} which is the initial value), \texttt{l}, \texttt{c} and \texttt{r} (when the user has used the corresponding key in the optional argument of the specifier \textit{X}).

\begin{verbatim}
\\str_set:Nn \l_@@_hpos_col_str { j }
\end{verbatim}

The possible values of \texttt{\l_@@_vpos_col_str} are \texttt{p} (the initial value), \texttt{m} and \texttt{b} (when the user has used the corresponding key in the optional argument of the specifier \textit{X}).

\begin{verbatim}
\\tl_set:Nn \l_@@_vpos_col_str { p }
\end{verbatim}

The integer \texttt{\l_@@_weight_int} will be the weight of the \textit{X} column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the \textit{X} columns are used in the computation of the actual width of those columns as in \texttt{tabu} of \texttt{tabularray}.

\begin{verbatim}
\\int_zero_new:N \l_@@_weight_int
\\int_set:Nn \l_@@_weight_int { 1 }
\\keys_set_known:nnN { WithArrows / p-column } { \#1 } \l_tmpa_tl
\\keys_set:nV { WithArrows / X-column } \l_tmpa_tl
\\int_compare:nNnT \l_@@_weight_int < 0
\{ \\@@_error:nx { negative~weight } { \int_use:N \l_@@_weight_int } \}
\\int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
\\int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int
\end{verbatim}

We test whether we know the width of the \textit{X}-columns by reading the \texttt{aux} file (after the first compilation, the width of the \textit{X}-columns is computed and written in the \texttt{aux} file).

\begin{verbatim}
\\bool_if:NTF \l_@@_X_columns_aux_bool
\{ \\@@_patch_preamble_iv_iv:nn
\{ \l_@@_weight_int \l_@@_X_columns_dim \}
\{ minipage \}
\}
\\tl_gput_right:Nn \g_@@_preamble_tl
\\\\{ \\@@_cell_begin:w
\\\bool_set_true:N \l_@@_X_column_bool
\}
\\tl_gput_right:Nn \g_@@_post_action_cell_tl
\\hbox_set:Nn \l_@@_cell_box { } \end{verbatim}

The following code will nullify the box of the cell.

\begin{verbatim}
\\tl_gput_right:Nn \g_@@_post_action_cell_tl
\{ \\hbox_set:Nn \l_@@_cell_box { } \}
\end{verbatim}
We put a \{minipage\} to give to the user the ability to put a command such as \centering in the RowStyle.

\begin{minipage}{5 cm} \arraybackslash \\
c
\end{minipage}
\@@_cell_end:
\int_gincr:N \c@jCol
\@@_patch_preamble_xi:n

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

The redefinition of \multicolumn

The following command must not be protected since it begins with \multispan (a TeX primitive).

The following lines are from the definition of \multicolumn in array (and not in standard LaTeX). The first line aims to raise an error if the user has put more that one column specifier in the preamble of \multicolumn.
You do the expansion of the (small) preamble with the tools of \texttt{array}.

\begin{verbatim}
\@temptokena = { #2 }
\@temptawtrue
\@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
\end{verbatim}

Now, we patch the (small) preamble as we have done with the main preamble of the array.

\begin{verbatim}
\tl_gclear:N \g_@@_preamble_tl
\exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
\end{verbatim}

The following lines are an adaptation of the definition of \texttt{multicolumn} in \texttt{array}.

\begin{verbatim}
\exp_args:NV \@mkpream \g_@@_preamble_tl
@addtopreamble \empty
\endgroup
\end{verbatim}

Now, you do a treatment specific to \texttt{nicematrix} which has no equivalent in the original definition of \texttt{multicolumn}.

\begin{verbatim}
\int_compare:nNnT { #1 } > 1
{ \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
  \{ \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } \} }
\seq_gput_right:Nx \g_@@_multicolumn_sizes_seq { \g_@@_pos_of_blocks_seq
  { \int_compare:nNnTF \c@jCol = 0
    { \int_eval:n { \c@iRow + 1 } }
    { \int_use:N \c@iRow }
  }
  { \int_eval:n { \c@jCol + #1 } }
  { } % for the name of the block
}
\end{verbatim}

The following lines were in the original definition of \texttt{multicolumn}.

\begin{verbatim}
\cs_set:Npn \@sharp { #3 }
\@arstrut
\@preamble
\null
\end{verbatim}

We add some lines.

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

\begin{verbatim}
\end{verbatim}

The following commands will patch the (small) preamble of the \texttt{multicolumn}. All those commands have a \texttt{m} in their name to recall that they deal with the redefinition of \texttt{multicolumn}.

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_m_preamble:n #1
{ \str_case:nnF { #1 } c { \@@_patch_m_preamble_i:n #1 }
  l { \@@_patch_m_preamble_i:n #1 }
  r { \@@_patch_m_preamble_i:n #1 }
  > { \@@_patch_m_preamble_iii:nn #1 }
}
\end{verbatim}
For c, l and r
\cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
\tl_gput_right:Nn \g_@@_preamble_tl
\begin{minipage}{\dim_eval:n { #1 }}
\mode_leave_vertical:
\arraybackslash
\vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
\end{minipage}
\@@_cell_end:
\end{minipage}
\@@_patch_m_preamble:n
\q_stop

We test for the presence of a <.
\@@_patch_m_preamble_x:n

For >, ! and @
\cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
\tl_gput_right:Nn \g_@@_preamble_tl
\begin{minipage}{\dim_eval:n { #2 }}
\begin{minipage}{\dim_eval:n { #1 }}
\end{minipage}
\end{minipage}
\@@_patch_m_preamble:n

For |
\cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
\tl_gput_right:Nn \g_@@_preamble_tl
\begin{minipage}{\dim_eval:n { #1 }}
\end{minipage}
\@@_patch_m_preamble:n

For p, m and b
\cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn t #1
\tl_gput_right:Nn \g_@@_preamble_tl
\begin{minipage}{\dim_eval:n { #3 }}
\begin{minipage}{\dim_eval:n { #2 }}
\end{minipage}
\end{minipage}
\@@_patch_m_preamble:n

We test for the presence of a <.
\@@_patch_m_preamble_x:n
For \( w \) and \( W \)

\begin{verbatim}
\cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn \#1 \#2 \#3 \#4
  \tl_gput_right:Nn \g_@@_preamble_tl
  { \hbox_set:Nw \l_@@_cell_box
    \@@_cell_begin:w
    \str_set:Nn \l_@@_hpos_cell_str \#3
  }
  c
  < \{
    \@@_cell_end:
    \#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 test for the presence of a \(<\).

\begin{verbatim}
\@@_patch_m_preamble_x:n \#1
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@_put_box_in_flow:
  \box_set_ht:Nn \l_tmpa_box \box_ht:N \l_tmpa_box + \l_tmpa_dim
  \box_set_dp:Nn \l_tmpa_box \box_dp:N \l_tmpa_box + \l_tmpb_dim
  \tl_if_eq:NnTF \l_@@_baseline_tl \text{c}
  \exp_args:NNx
    \clist_if_in:NnT \l_@@_vlines_clist \c@jCol + 1
    \tl_gput_right:Nn \g_@@_preamble_tl
    \! \{ \skip_horizontal:N \arrayrulewidth \}
\end{verbatim}

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

\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 - \int_eval:n { \c@iRow + 1 } }
  \dim_gadd:NN \g_tmpa_dim \pgf@y
  \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }
\endpgfpicture

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

\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 }
}{
  \str_case:VnF \l_@@_baseline_tl {
    t { \int_set:Nn \l_tmpa_int 1 }
    b { \int_set_eq:NN \l_tmpa_int \c@iRow }
  }
  \\bool_lazy_or:nnT
    { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
    { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
  {
    \@@_error:n { bad-value-for-baseline }
    \int_set:Nn \l_tmpa_int 1
  }
  \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
}
\endpgfpicture

Now, \g_tmpa_dim contains the value of the y translation we have to to.

\dim_gsub:NN \g_tmpa_dim \fontdimen22 \textfont2
\dim_gsub:NN \g_tmpa_dim \pgf@y

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:
{
  With an environment \{Matrix\}, you want to remove the exterior \arraycolsep but we don’t know the number of columns (since there is no preamble) and that’s why we can’t put {} at the end of the preamble. That’s why we remove a \arraycolsep now.
}
\bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
\{
  \box_set_wd:Nn \l_@@_the_array_box
  \{ \box_wd:N \l_@@_the_array_box - \arraycolsep \}
\}

We need a \texttt{\{minipage\}} because we will insert a LaTeX list for the tabular notes (that means that a \texttt{\vtop{\hsize=\ldots}} is not enough).
\begin{minipage}{\textwidth}
\hbox
\box_use_drop:N \l_@@_the_array_box

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

We don’t do the following test with \texttt{\c@tabularnote} because the value of that counter is not reliable when the command \texttt{\ttabbox} of \texttt{floatrow} is used (because \texttt{\ttabbox} de-activate \texttt{\stepcounter} because if compiles several twice its tabular).
\begin{enumerate}
\end{enumerate}

The TeX group is for potential specifications in the \texttt{\l_@@_notes_code_before_tl}.
\begin{enumerate}
\end{enumerate}

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

The following \texttt{\par} is mandatory for the event that the user has put \texttt{\footnotesize} (for example) in the notes/code-before.
\begin{enumerate}
\end{enumerate}
The two dimensions `\aboverulesep` et `\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_tmpa_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 \texttt{light-syntax} is in force or not). When the key \texttt{light-syntax} is not used, the construction is a standard environment (and, thus, it’s possible to use verbatim in the array).

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

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

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

Now, you extract the \texttt{CodeAfter} of the body of the environment. Maybe, there is no command \texttt{CodeAfter} in the body. That’s why you put a marker \texttt{CodeAfter} after \texttt{#1}. If there is yet a \texttt{CodeAfter} in \texttt{#1}, this second (or third...) \texttt{CodeAfter} will be caught in the value of \texttt{g_nicematrix_code_after_tl}. That doesn’t matter because \texttt{CodeAfter} will be set to \texttt{no-op} before the execution of \texttt{g_nicematrix_code_after_tl}.

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

The body of the array, which is stored in the argument \texttt{#1}, is now splitted into items (and \texttt{not} tokens).
If the environment uses the option `last-row` without value (i.e., without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

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

Here is the call to `array` (we have a dedicated macro `\@@_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 affection because, when executing `\l_tmpa_tl`, we will exit the first cell of the array.

\begin{verbatim}
\seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
\@@_line_with_light_syntax_i:V \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}

We reprint in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

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

The command `\@@_create_col_nodes`: will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specifies 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
{ \bool_if:NT \l_@@_code_before_bool
{ \pgfsys@markposition { \@@_env: - col - 0 } }
\pgfpicture
\pgfreememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
\endpgfpicture
\end{verbatim}
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@tmpaSkip 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@tmpaSkip, like all the affectations, must be done after the \omit of the cell.

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

\skip_gset:Nn \g@tmpaSkip { 0 pt plus 1 fill }
\bool_if:NF \l@@auto_columns_width_bool {
  \dim_compare:nNnT \l@@columns_width_dim > \c_zero_dim }
\bool_lazy_and:nnTF \l@@auto_columns_width_bool \l@@block_auto_columns_width_bool {
  \skip_gset_eq:NN \g@tmpaSkip \g@@max_cell_width_dim
}
We begin a loop over the columns. The integer \texttt{$\gtmpa_int$} will be the number of the current column. This integer is used for the Tikz nodes.

\begin{verbatim}
\int_gset:Nn \gtmpa_int 1
\bool_if:NTF \g_@@_last_col_found_bool
{ \prg_replicate:nn \int_max:nn \g_@@_col_total_int - 3 \g_@@_col_total_int - 2
} \prg_replicate:nn \int_max:nn \g_@@_col_total_int - 2
{ \omit \int_gincr:N \gtmpa_int
\skip_horizontal:N \gtmpa_skip
\bool_if:NT \l_@@_code_before_bool
{ \hbox \\
\skip_horizontal:N -0.5\arrayrulewidth \pgfsys@markposition \l_@@_env: - col - 2 } \skip_horizontal:N 0.5\arrayrulewidth
}
\endpgfpicture
\end{verbatim}

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

\begin{verbatim}
\skip_horizontal:N \gtmpa_skip
\bool_if:NT \l_@@_code_before_bool
{ \hbox \\
\skip_horizontal:N -0.5\arrayrulewidth \pgfsys@markposition \l_@@_env: - col - \int_eval:n \gtmpa_int + 1 } \skip_horizontal:N 0.5\arrayrulewidth
\end{verbatim}

We create the \texttt{col} node on the right of the current column.

\begin{verbatim}
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate \l_@@_env: - col - \int_eval:n \gtmpa_int + 1
\pgfpoint -0.5\arrayrulewidth \c_zero_dim
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias \l_@@_name_str - col - \int_eval:n \gtmpa_int + 1 \l_@@_env: - col - 2
} \endpgfpicture
\end{verbatim}

\&\omit
The two following lines have been added on 2021-12-15 to solve a bug mentioned by Joao Luis Soares by mail.

\int_compare:nNnT \g_@@_col_total_int = 1
\{ \skip_gset:Nn \g_tmpa_skip { 0 pt plus 1 fill } \}
\skip_horizontal:N \g_tmpa_skip
\int_gincr:N \g_tmpa_int
\bool_lazy_all:nT
\{ \l_@@_NiceArray_bool
\{ \bool_not_p:n \l_@@_NiceTabular_bool \}
\{ \clist_if_empty_p:N \l_@@_vlines_clist \}
\{ ! \l_@@_bar_at_end_of_pream_bool \}
\}
\{ \skip_horizontal:N -\col@sep \}
\bool_if:NT \l_@@_code_before_bool
{ \hbox
\{ \skip_horizontal:N -0.5\arrayrulewidth
With an environment \{Matrix\}, you want to remove the exterior \arraycolsep but we don't know the number of columns (since there is no preamble) and that's why we can't put @{} at the end of the preamble. That's why we remove a \arraycolsep now.
\bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
{ \skip_horizontal:N -\arraycolsep \}
\pgfsys@markposition
{ \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
\skip_horizontal:N 0.5\arrayrulewidth
\bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
{ \skip_horizontal:N \arraycolsep }
\pgfpicture
\pgfrememberpictureposition
\pgfcoordinate { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
\bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
{ \pgfpaint
\pgfpoint { -0.5 \arrayrulewidth - \arraycolsep } \c_zero_dim
\}
\pgfpoint { -0.5 \arrayrulewidth } \c_zero_dim
\}
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias
\{ \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } \}
\{ \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } \}
\}
\endpgfpicture

\bool_if:NT \g_@@_last_col_found_bool
{ \hbox_overlap_right:n
\{ \skip_horizontal:N \g_@@_width_last_col_dim
\bool_if:NT \l_@@_code_before_bool
{ \pgfsys@markposition
\{ \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } \}
}
Here is the preamble for the “first column” (if the user uses the key \first-col)
\tl_const:Nn \c_@@_preamble_first_col_tl
{
> 
}

At the beginning of the cell, we link \CodeAfter to a command which do begins with \ (whereas the standard version of \CodeAfter begins does not).
\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
\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 }
}
\{
\l_@@_code_for_first_col_tl
\xglobal \colorlet { nicematrix-first-col } { . }
}

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.
1
<
{
\@@_math_toggle_token:
\hbox_set_end:
\bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
\@@_adjust_size_box:
\@@_update_for_first_and_last_row:
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_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
{ >
}

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

\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

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

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

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

\hbox_set:Nw \l_@@_cell_box
\@@_math_toggle_token:
\bool_if:NT \l_@@_small_bool \scriptstyle

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

\int_compare: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 } { . } }
}
1 <

{ \@@_math_toggle_token:
\hbox_set_end:
\bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
\@@_adjust_size_box:
\@@_update_for_first_and_last_row:
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_cell:\}
\} \bool_gset_false:N \g_@@_empty_cell_bool
\}

The environment \{NiceArray\} is constructed upon the environment \{NiceArrayWithDelims\} but, in fact, there is a flag \l_@@_NiceArray_bool. In \{NiceArrayWithDelims\}, some special code will be executed if this flag is raised.
\NewDocumentEnvironment { NiceArray } { }
\{ \bool_set_true:N \l_@@_NiceArray_bool
\str_if_empty:NT \g_@@_name_env_str { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
\@@_test_if_math_mode:
NiceArrayWithDelims . .
\}
\{ \endNiceArrayWithDelims \}

We create the variants of the environment \{NiceArrayWithDelims\}.
\cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
\{ \NewDocumentEnvironment { \#1 NiceArray } { }
\{ \str_if_empty:NT \g_@@_name_env_str { \str_gset:Nn \g_@@_name_env_str { \#1 NiceArray } }
\@@_test_if_math_mode:
\NiceArrayWithDelims \#2 \#3
\}
\{ \endNiceArrayWithDelims \}
\}
\@@_def_env:nnn p ( )
\@@_def_env:nnn b [ ]
\@@_def_env:nnn B \{ \}
\@@_def_env:nnn v | |
\@@_def_env:nnn V \| |
The environment \{NiceMatrix\} and its variants

\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
\bool_set_true:N \l_@@_Matrix_bool
\use:c { #1 NiceArray }
\*
\int_compare:nNnTF \l_@@_last_col_int < 0 \c@MaxMatrixCols
 { \int_eval:n { \l_@@_last_col_int - 1 } }
{ > \@@_cell_begin:w #2 < \@@_cell_end: }
\}
\cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n e }
\clist_map_inline:nn { { } , p , b , B , v , V }
\NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
{ \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
\tl_set:Nn \l_@@_type_of_col_tl c
\keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
\@@_begin_of_NiceMatrix:ne { #1 } \l_@@_type_of_col_tl
}{ \use:c { end #1 NiceArray } }

The following command will be linked to \NotEmpty in the environments of nicematrix.
\cs_new_protected:Npn \@@_NotEmpty: 
{ \bool_gset_true:N \g_@@_not_empty_cell_bool }

\{NiceTabular\}, \{NiceTabularX\} and \{NiceTabular*\}

\NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
{ \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
{ \dim_set_eq:NN \l_@@_width_dim \linewidth }
\str_gset:Nn \g_@@_name_env_str { NiceTabular }
\keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
\bool_set_true:N \l_@@_NiceTabular_bool
\NiceArray { #2 }
}{ \endNiceArray }
\cs_set_protected:Npn \@@_newcolumntype #1
{ \cs_if_free:cT { NC @ find @ #1 } { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
\cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
\peek_meaning:NTF [ \newcol@ #1 ]
{ \newcol@ #1 [ 0 ] }
}

\NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }

The following code prevents the expansion of the 'X' columns with the definition of that columns in `tabularx` (this would result in an error in `{NiceTabularX}`).

```latex
\bool_if:NT \c_@@_tabularx_loaded_bool \{ \newcolumntype { X } \{ \@@_X \} \\
\str_gset:Nn \g_@@_name_env_str \{ \texttt{NiceTabularX} \} \\
\dim_zero_new:N \l_@@_width_dim \\
\dim_set:Nn \l_@@_width_dim \{ #1 \} \\
\keys_set:nn \{ \texttt{NiceMatrix} / \texttt{NiceTabular} \} \{ #2 , #4 \} \\
\bool_set_true:N \l_@@_NiceTabular_bool \\
\NiceArray \{ #3 \} \\
\} \\
\endNiceArray \\
\NewDocumentEnvironment \{ \texttt{NiceTabular*} \} \{ m O \{ \} m O \{ \} \} \\
\{ \\
\str_gset:Nn \g_@@_name_env_str \{ \texttt{NiceTabular*} \} \\
\dim_set:Nn \l_@@_tabular_width_dim \{ #1 \} \\
\keys_set:nn \{ \texttt{NiceMatrix} / \texttt{NiceTabular} \} \{ #2 , #4 \} \\
\bool_set_true:N \l_@@_NiceTabular_bool \\
\NiceArray \{ #3 \} \\
\} \\
\endNiceArray \\
\cs_new_protected:Npn \@@_after_array: \\
\group_begin: \\
\boolean_if:NT \g_@@_last_col_found_bool \{ \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int \} \\
\bool_if:NT \l_@@_last_col_without_value_bool \{ \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int \} \\
\tl_gput_right:Nx \g_@@_aux_tl \\
\seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq \\
\bool_if_empty:NF \g_@@_pos_of_blocks_seq \{ \exp_not:N \g_@@_size_seq \\
\int_use:N \l_@@_first_row_int , \\
\int_use:N \c@iRow , \\
\int_use:N \g_@@_row_total_int , \\
\int_use:N \l_@@_first_col_int , \\
\int_use:N \c@jCol , \\
\int_use:N \g_@@_col_total_int \\
\} \\
\seq_if_empty:NF \g_@@_pos_of_blocks_seq \\
\endgroup \\
\endcsname
```

After the construction of the array

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.

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

It’s also time to give to \l_@@_last_row_int its real value.

We write also the potential content of \g_@@_pos_of_blocks_seq. It will be used to recreate the blocks with a name in the \CodeBefore and also if the command \rowcolors is used with the key `respect-blocks`).

```latex
\seq_if_empty:NF \g_@@_pos_of_blocks_seq 
\endcsname
\endcsname
```

133
\tl_gput_right:Nx \g_@@_aux_tl
\seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
\seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }

\seq_if_empty:NF \g_@@_multicolumn_cells_seq
\tl_gput_right:Nx \g_@@_aux_tl
\seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
\seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
\seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
\seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }

Now, you create the diagonal nodes by using the row nodes and the col nodes.

\@@_create_diag_nodes:

We create the aliases using last for the nodes of the cells in the last row and the last column.

\pgfpicture
\int_step_inline:nn \c@iRow
{ \pgfnodealias
{ \@@_env: - ##1 - last }
{ \@@_env: - ##1 - \int_use:N \c@jCol }
}
\int_step_inline:nn \c@jCol
{ \pgfnodealias
{ \@@_env: - last - ##1 }
{ \@@_env: - \int_use:N \c@iRow - ##1 }
}
\str_if_empty:NF \l_@@_name_str
\int_step_inline:nn \c@iRow
{ \pgfnodealias
{ \l_@@_name_str - ##1 - last }
{ \@@_env: - ##1 - \int_use:N \c@jCol }
}
\int_step_inline:nn \c@jCol
{ \pgfnodealias
{ \l_@@_name_str - last - ##1 }
{ \@@_env: - \int_use:N \c@iRow - ##1 }
}
\endpgfpicture

By default, the diagonal lines will be parallelized\textsuperscript{68}. 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

The dimensions \g_@@_delta_x_one_dim and \g_@@_delta_y_one_dim will contain the $\Delta_x$ and $\Delta_y$ of the first \Ddots diagonal. We have to store these values in order to draw the others \Ddots

\textsuperscript{68}It’s possible to use the option parallelize-diags to disable this parallelization.
If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.
\begin{verbatim}
\bool_if:NT \l_@@_small_bool {
\dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
\dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }
}
\end{verbatim}
The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_start_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.
\begin{verbatim}
\dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.6 \l_@@_xdots_shorten_start_dim }
\dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.6 \l_@@_xdots_shorten_end_dim }
\end{verbatim}

Now, we actually draw the dotted lines (specified by `\dots`, `\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 won’t do anything. The corners are computed in `\l_@@_corners_cells_seq` which will contain all the cells which are empty (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 `\Block{1-}`).
\begin{verbatim}
\@@_adjust_pos_of_blocks_seq:
\end{verbatim}

Now, the internal `code-after` and then, the `\CodeAfter`.
\begin{verbatim}
\bool_if:NT \c_@@_tikz_loaded_bool {
\tikzset {
\textbf{every-picture} / .style =
\\{ 
\textbf{overlay} ,
\textbf{remember-picture} ,
\textbf{name-prefix} = \@@_env: -
\}\}
\end{verbatim}

135
When *light-syntax* is used, we insert systematically a \CodeAfter in the flow. Thus, it’s possible to have two instructions \CodeAfter and the second may be in \g_nicematrix_code_after_tl. That’s why we set Code-after to be no-op now.

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

And here’s the \CodeAfter. Since the \CodeAfter may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command \@@_CodeAfter_keys:

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

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

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 colortbl. But we want also to be able to use color for the rules of the array when colortbl is not loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by colortbl.

The following command will extract the potential options (between square brackets) at the beginning of the \CodeAfter (that is to say, when \CodeAfter is used) and treat the options of that “command” \CodeAfter. Idem for the \CodeBefore.

We remind that the first mandatory argument of the command \Block is the size of the block with the special format \text{i-j}. However, the user is allowed to omit \text{i} or \text{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

\footnote{e.g. \text{\color{rgb}(0.5,0.5,0)}}
\_@@\_pos\_of\_blocks\_seq (and \_@@\_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).

The following command must not be protected.

```latex
\cs_new:Npn \@@_adjust_pos_of_blocks_seq:
{ \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq_i:nnnnn \#1 }
```

The following command must be protected.

```latex
\cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
{ #1 #2
\int_compare:nNnTF { #3 } > { 99 } { \int_use:N \c@iRow } { #3 }
\int_compare:nNnTF { #4 } > { 99 } { \int_use:N \c@jCol } { #4 }
#5 }
```

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

```latex
\hook_gput_code:nnn { begindocument } { . }
\cs_new_protected:Npx \@@_draw_dotted_lines:
{ \c_@@_pgfortikzpicture_tl \@@_draw_dotted_lines_i: \c_@@_endpgfortikzpicture_tl }
```

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

```latex
\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 }
```

We define a new PGF shape for the diag nodes because we want to provide a anchor called .5 for those nodes.

```latex
\pgfdeclareshape { @@\_diag\_node }
```
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).

\cs_new_protected:Npn \@@_create_diag_nodes:
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
\@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
\dim_set_eq:NN \l_tmpa_dim \pgf@x
\@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
\dim_set_eq:NN \l_tmpb_dim \pgf@y
\@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
\dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
\@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
\dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
\pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
\now, \l_tmpa_dim \text{ and } \l_tmpb_dim \text{ become the width and the height of the node (of shape } \@à_diag_node \text{) that we will construct.}
\dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
\dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
\pgfnode { \@à_diag_node } { center } { } { \@@_env: - ##1 } { }
\str_if_empty:NF \l_@@_name_str { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
\endpgfpicture

Now, \_l_tmpa_dim \text{ and } \_l_tmpb_dim \text{ become the width and the height of the node (of shape } \@à_diag_node \text{) that we will construct.}
\int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
\@@_qpoint:n { row - \int_min:nn { \_l_tmpa_int } { \c@iRow + 1 } }
\dim_set_eq:NN \_l_temp_dim \pgf@y
\@@_qpoint:n { col - \int_min:nn { \_l_tmpa_int } { \c@jRow + 1 } }
\pgfcoordinate { \_l_@@_name_str - \_l_tmpa_int } { \pgfpoint \pgf@x \_l_tmpb_dim }
\pgfnodealias { \_l_@@_name_str - last } { \_l_@@_name_str - \_l_tmpa_int }
\str_if_empty:NF \l_@@_name_str { \pgfnodealias { \l_@@_name_str - last } { \_l_@@_name_str - last } }
\endpgfpicture

Now, the last node. Of course, that is only a coordinate because there is not .5 anchor for that node.
\int_set:Nn \_l_temp_int { \int_max:nn \c@iRow \c@jCol + 1 }
\dim_set_eq:NN \_l_temp_dim \pgf@y
\@@_qpoint:n { row - \int_min:nn { \_l_temp_int } { \c@iRow + 1 } }
\dim_set_eq:NN \_l_temp_dim \pgf@y
\pgfcoordinate { \_l_@@_name_str - \_l_temp_int } { \pgfpoint \pgf@x \_l_tmpa_dim }
\pgfnodealias { \_l_@@_name_str - last } { \_l_@@_name_str - \_l_temp_int }
\str_if_empty:NF \_l_@@_name_str { \pgfnodealias { \_l_@@_name_str - last } { \_l_@@_name_str - last } }
\endpgfpicture

We draw the dotted lines

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

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the \(x\)-value of the orientation vector of the line;
- the fourth argument is the \(y\)-value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```latex
3277 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn \#1 \#2 \#3 \#4
3278 {
3279 \cs_set:cpn { \@@_dotted _ \#1 - \#2 } { }
3280 \cs_set:Nn \l_@@_initial_i_int { \#1 }
3281 \cs_set:Nn \l_@@_initial_j_int { \#2 }
3282 \cs_set:Nn \l_@@_final_i_int { \#1 }
3283 \cs_set:Nn \l_@@_final_j_int { \#2 }
3284 \bool_set_false:N \l_@@_stop_loop_bool
3285 \bool_do_until:Nn \l_@@_stop_loop_bool
3286 {
3287 \int_add:Nn \l_@@_final_i_int { \#3 }
3288 \int_add:Nn \l_@@_final_j_int { \#4 }
3289 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3290 { \int_compare:nNnT \l_@@_final_j_int = 1
3291 { \bool_set_true:N \l_@@_final_open_bool }
3292 \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3293 { \bool_set_true:N \l_@@_final_open_bool }
3294 }
3295 \int_compare:nNnTF \l_@@_final_i_int < \l_@@_col_min_int
3296 { \int_compare:nNnT \l_@@_final_j_int = -1
3297 { \bool_set_true:N \l_@@_final_open_bool }
3298 }
3299 }
```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

Initialization of variables.

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.

We test if we are still in the matrix.

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

139
\int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_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
{
@@_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
}

\cs_if_exist:cTF
{
pgf@sh@ns@\@@_env:-\int_use:N \l_@@_final_i_int - \int_use:N \l_@@_final_j_int
}
\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.
{
\cs_set:cpn
{
@@_dotted_/\int_use:N \l_@@_final_i_int - \int_use:N \l_@@_final_j_int
}
}

For \l_@@_initial_i_int and \l_@@_initial_j_int the programmation is similar to the previous one.
\bool_set_false:N \l_@@_stop_loop_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
{\int_sub:Nn \l_@@_initial_i_int { #3 }
\int_sub:Nn \l_@@_initial_j_int { #4 }
\bool_set_false:N \l_@@_initial_open_bool
\int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
{\int_compare:nNnTF { #3 } = 1
{ \bool_set_true:N \l_@@_initial_open_bool }
{ \int_compare:nNnT \l_@@_initial_j_int = \l_@@_col_min_int -1
{ \bool_set_true:N \l_@@_initial_open_bool }
}
\int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
{ \int_compare:nNnT { #4 } = 1
{ \bool_set_true:N \l_@@_initial_open_bool }
{ \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
{ \int_compare:nNnT { #4 } = -1
{ \bool_set_true:N \l_@@_initial_open_bool }
}
\bool_if:NTF \l_@@_initial_open_bool
{ \int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool }
{ \cs_if_exist:cTF
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { \int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_stop_loop_bool }
{ \cs_if_exist:cTF
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { \cs_set:cpn
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { } } }

\bool_if:NTF \l_@@_initial_open_bool
{ \int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool }
{ \cs_if_exist:cTF
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { \int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_stop_loop_bool }
{ \cs_if_exist:cTF
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { \cs_set:cpn
{ \int_use:N \l_@@_initial_i_int -\int_use:N \l_@@_initial_j_int
} { } } }

\cs_set:cnpn
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
\seq_map_inline:Nn \g_@@_submatrix_seq
\bool_if:nT { \int_compare_p:n { #3 <= #1 } 
\& \int_compare_p:n { #1 <= #5 } 
\& \int_compare_p:n { #4 <= #2 } 
\& \int_compare_p:n { #2 <= #6 } }

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{ \bool_if:nT { \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_set_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:
{ }

#1 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.

The following command (\textit{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 #1 and column #2. As of now, it's only the whole array (excepted exterior rows and columns).

\cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
{ }
\cs_set_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
{ }
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnn #1 #2 #3
{ }
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnn #1 #2 #3 #4
{ }
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5
{ }

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnnn #1 #2 #3 #4 #5 #6
{ }

\cs_set_protected:Npn \@@_set_initial_coords:
{ }
\cs_set_protected:Npn \@@_set_final_coords:
{ }

We do a loop over all the submatrices specified in the \texttt{code-before}. We have stored the position of all those submatrices in \g_@@_submatrix_seq.

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{ }
\cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
{ }

\cs_new_protected:Npn \@@_set_initial_coords:
{ }
\cs_new_protected:Npn \@@_set_final_coords:
{ }

\seq_map_inline:Nn \g_@@_submatrix_seq
\bool_if:nT { \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 } }

\bool_if:nT { \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 } }

\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 } }

\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 } }

\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 } }

\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 } }

142
If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

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

\begin{verbatim}
  \cs_if_exist:cT
  \{ \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int
  \{ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int
  \{ \@@_env: - \int_use:N \l_@@_initial_i_int
  \{ \@@_env: - \int_use:N \l_@@_initial_j_int
  \}
  \}
\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_final_dim = \{- \c_max_dim \}
\@@_qpoint:n { col - \int_eval:n \{ \l_@@_final_j_int + 1 \} }
\dim_set_eq:NN \l_@@_x_final_dim \pgf@x
\dim_sub:Nn \l_@@_x_final_dim \col@sep

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

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.
\group_begin:
\int_compare:nNnTF { #1 } = 0
{ \color { nicematrix-first-row } }
\int_compare:nNnT { #1 } = \l_@@_last_row_int
{ \color { nicematrix-last-row } }
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Ldots:
\group_end:

The command \@@_actually_draw_Ldots: has the following implicit arguments:
• \l_@@_initial_i_int
• \l_@@_initial_j_int
• \l_@@_initial_open_bool
• \l_@@_final_i_int
• \l_@@_final_j_int
• \l_@@_final_open_bool.

The following function is also used by \Hdotsfor.
\cs_new_protected:Npn \@@_actually_draw_Ldots:
{ \bool_if:NTF \l_@@_initial_open_bool
{ \@@_open_x_initial_dim:
\@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
}
{ \@@_set_initial_coords_from_anchor:n { base-east } }
\bool_if:NTF \l_@@_final_open_bool
We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option line-style is used (?).

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

\cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
\@@_adjust_to_submatrix:nn { #1 } { #2 }
\cs_if_free:cT { @@ _ dotted _ #1 - #2 }
\begin{itemize}
\item \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.
\end{itemize}

\bool_lazy_and:nnTF 145
\group_begin:
\int_compare:nNnTF { #1 } = \l_@@_last_row_int
{ \color { nicematrix-last-row } }
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:

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

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

\cs_new_protected:Npn \@@_actually_draw_Cdots:
\bool_if:NTF \l_@@_initial_open_bool
{ \@@_open_x_initial_dim: }
\{ \@@_set_initial_coords_from_anchor:n { mid-east } }
\bool_if:NTF \l_@@_final_open_bool
{ \@@_open_x_final_dim: }
\{ \@@_set_final_coords_from_anchor:n { mid-west } }
\bool_lazy_and:nnTF
The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\group_begin:
  \int_compare:nNnTF { #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:
  {
    \bool_set_false:N \l_tmpa_bool 
    First the case when the line is closed on both ends.
    \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool 
    { 
      \@@_set_initial_coords_from_anchor:n { south\~west } 
      \@@_set_final_coords_from_anchor:n { north\~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 c or may be considered as if.
    \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 
    \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }  
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x 
    \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }  
    \dim_set:NN \l_@@_x_initial_dim \l_@@_x_initial_dim \pgf@x 
    \dim_set:NN \l_@@_x_final_dim \l_@@_x_final_dim \pgf@x 
  }
We may think that the final user won’t use a "last column" which contains only a command \Vdots. However, if the \Vdots is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

\int_compare:nNnT \l_@@_last_col_int > {-2}
\int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
\dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
\dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
\dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
\dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
\dim_set_eq:NN \l_@@_right_margin_dim \l_tmpa_dim
\dim_add:Nn \l_@@_extra_right_margin_dim \l_tmpa_dim
\dim_add:Nn \l_@@_x_final_dim \l_@@_x_initial_dim
\dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
\bool_if:NTF \l_@@_final_open_bool
\dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
\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
\dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
\@@_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:nnnn { #1 } { #2 } 1 1

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.
\group_begin:
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:VF \l_@@_xdots_color_tl \color { \l_@@_xdots_color_tl }
\@@_actually_draw_Ddots:
\group_end:

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

- \l_@@_initial_i_int
\cs_new_protected:Npn \@@_actually_draw_Ddots: 
{
\bool_if:NTF \l_@@_initial_open_bool 
 {
\@@_open_y_initial_dim: 
\@@_open_x_initial_dim: 
}
\bool_if:NTF \l_@@_final_open_bool 
 {
\@@_open_x_final_dim: 
\dim_set_eq:NN \l_@@_x_final_dim \pgf@x 
}
\bool_if:NTF \l_@@_final_coords_from_anchor:n { north-west } 
\@@_set_final_coords_from_anchor:n 
}
\dim_gset:Nn \g_@@_delta_x_one_dim 
\l_@@_y_final_dim - \l_@@_y_initial_dim }
\dim_gset:Nn \g_@@_delta_y_one_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 
}
\@@_draw_line: 
}
\bool_if:NTF \l_@@_parallelize_diags_bool 
{ 
\int_gincr:N \g_@@_ddots_int 
We test if the diagonal line is the first one (the counter \g_@@_ddots_int is created for this usage). 
\int_compare:nNnTF \g_@@_ddots_int = 1 
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.
\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 }
\dim_gset:Nn \g_@@_delta_y_one_dim 
\g_@@_delta_x_one_dim 
\dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim 
}
\dim_set:Nn \l_@@_y_final_dim 
( \l_@@_x_final_dim - \l_@@_x_initial_dim ) * 
\dim_set:Nn \l_@@_y_final_dim 
\l_@@_y_initial_dim + 
\l_@@_x_final_dim + 
\dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim 
}
\@@_draw_line: 
}

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 

We draw the \Iddots diagonals in the same way. 
The first and the second arguments are the coordinates of the cell where the command has been 
issued. The third argument is the list of the options. 
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3 
{ 
\@@_adjust_to_submatrix:nn { #1 } { #2 } 
\cs_if_free:cT { \00 _ _ dotted _ #1 - #2 } 
}
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

The command `\@@_actually_draw_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.

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.

The command `\@@_actually_draw_Iddots:` has the following implicit arguments:
The actual instructions for drawing the dotted lines 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: #1 #2 #3 #4 #5 #6
{
  \begin {scope}
    \@@_draw_unstandard_dotted_line:o { #1 , #2 , #3 , #4 , #5 , #6 }
  \end {scope}
}
```

The boolean `\l_@@_dotted_bool` is raised for the rules specified by either `\hdottedline` or : (or the letter specified by `letter-for-dotted-lines`) in the preamble of the array.

```
\begin {scope}
  \@@_draw_unstandard_dotted_line:o { #1 , #2 , #3 , #4 , #5 , #6 }
\end {scope}
```

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

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1 #2 #3
{
  \draw
  [ #1 ,
    shorten-> = \l_@@_xdots_shorten_end_dim ,
    shorten<- = \l_@@_xdots_shorten_start_dim ,
  ]
  ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
  -- node \[ sloped , above \] { $ \scriptstyle #2 $ }
    node \[ sloped , below \] { $ \scriptstyle #3 $ }
  ( \l_@@_x_final_dim , \l_@@_y_final_dim )
}
```

Be careful: We can’t put `\c_math_toggle_token` instead of $ in the following lines because we are in the contents of Tikz nodes (and they will be rescanned if the Tikz library babel is loaded).
The command \@@_draw_standard_dotted_line: draws the line with our system of dots (which gives a dotted line with real round dots).

The dimension \l_@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.
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:
\group_end:
\dim_const:Nn \c_@@_max_l_dim { 50 \text{ cm} }
\cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
\{ 
The number of dots will be \l_tmpa_int + 1.
\bool_if:NTF \l_@@_initial_open_bool
\{ 
\bool_if:NTF \l_@@_final_open_bool
\{ 
\int_set:Nn \l_tmpa_int
\{ \dim_ratio:nn \l_@@_l_dim \l_@@_xdots_inter_dim \}
\}
\}
\int_set:Nn \l_tmpa_int
\{ \dim_ratio:nn
\l_@@_l_dim - \l_@@_xdots_shorten_start_dim \}
\l_@@_xdots_inter_dim
\}
\}
\bool_if:NTF \l_@@_final_open_bool
\{ 
\int_set:Nn \l_tmpa_int
\{ \dim_ratio:nn
\l_@@_l_dim - \l_@@_xdots_shorten_end_dim \}
\l_@@_xdots_inter_dim
\}
\}
\int_set:Nn \l_tmpa_int
\{ \dim_ratio:nn
\l_@@_l_dim - \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim \}
\l_@@_xdots_inter_dim
\}
\}
The dimensions \l_tmpa_dim and \l_tmpb_dim are the coordinates of the vector between two dots in the dotted line.
In the loop over the dots, the dimensions \l_@@_x_initial_dim and \l_@@_y_initial_dim will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

\dim_gadd:Nn \l_@@_x_initial_dim
\dim_ratio:nn
\dim_gadd:Nn \l_@@_y_initial_dim
\dim_ratio:nn
\pgf@relevantforpicturesizefalse
\int_step_inline:nnn 0 \l_tmpa_int
{\pgfpathcircle {\pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
\dim_add:Nn \l_@@_x_initial_dim \l_tmpa_int
\dim_add:Nn \l_@@_y_initial_dim \l_tmpa_int
}\pgfusepathqfill

User commands available in the new environments

The commands \@@_Ldots, \@@_Cdots, \@@_Vdots, \@@_Ddots and \@@_Iddots will be linked to \Ldots, \Cdots, \Vdots, \Ddots and \Iddots in the environments \{NiceArray\} (the other environments of nicematrix rely upon \{NiceArray\}).

The syntax of these commands uses the character _ as embellishment and thats’ 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 \hook_gput_code:nnn \{ begindocument \} \{ . \} and the arg spec will be rescanned.
\exp_args:NNV \NewDocumentCommand \@_Ldots \l_@@_argspec_tl
{ \int_compare:nNnTF \c@jCol = 0
  { \@@_error:nn { in-first-col } \Ldots }
  { \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
    { \@@_error:nn { in-last-col } \Ldots }
    { \@@_instruction_of_type:nnn \c_false_bool { \Ldots }
      { #1 , down = #2 , up = #3 }
    }
  }
\bool_if:NF \l_@@_nullify_dots_bool
  { \phantom { \ensuremath { \@@_old_ldots } } }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@_Cdots \l_@@_argspec_tl
{ \int_compare:nNnTF \c@jCol = 0
  { \@@_error:nn { in-first-col } \Cdots }
  { \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
    { \@@_error:nn { in-last-col } \Cdots }
    { \@@_instruction_of_type:nnn \c_false_bool { \Cdots }
      { #1 , down = #2 , up = #3 }
    }
  }
\bool_if:NF \l_@@_nullify_dots_bool
  { \phantom { \ensuremath { \@@_old_cdots } } }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@_Vdots \l_@@_argspec_tl
{ \int_compare:nNnTF \c@iRow = 0
  { \@@_error:nn { in-first-row } \Vdots }
  { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
    { \@@_error:nn { in-last-row } \Vdots }
    { \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
      { #1 , down = #2 , up = #3 }
    }
  }
\bool_if:NF \l_@@_nullify_dots_bool
  { \phantom { \ensuremath { \@@_old_vdots } } }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@_Ddots \l_@@_argspec_tl
{ \int_case:nnF \c@iRow
  { 0 { \@@_error:nn { in-first-row } \Ddots }
    \l_@@_last_row_int { \@@_error:nn { in-last-row } \Ddots }
  }
  { \int_case:nnF \c@jCol

155
Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

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

In the environments of nicematrix, the command \multicolumn is redefined. We will patch the environment {tabular} to go back to the previous value of \multicolumn.

\cs_set_eq:NN \@@_old_multicolumn \multicolumn
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.

\begin{verbatim}
\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
\}
\}
\cs_new_protected:Npn \@@_Hdotsfor_i
\{
  \bool_set_false:N \l_@@_initial_open_bool
  \bool_set_false:N \l_@@_final_open_bool
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
  \int_compare:nNnTF { #2 } = 1
  { \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } } }
\}
\end{verbatim}

We don’t put \! before the last optional argument for homogeneity with \Cdots, etc. which have only one optional argument.

\begin{verbatim}
\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{
  \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
  { \@@_Hdotsfor:nnnn #1 #2 #3 #4 }

  \tl_set:NNn \g_@@_argspec_tl { #2 - 1 } \multicolumn { 1 } { c } { }
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_Hdotsfor:nnnn \l_@@_argspec_tl
\{
  \bool_lazy_and:nnTF
  \{ \int_compare_p:nNn \c@jRow = 0 \}
  \{ \int_compare_p:nNn \l_@@_first_row_int = 0 \}
  \{
    \bool_if:NTF \g_@@_after_row_zero_bool
    {
      \multicolumn { \l_@@_first_row_int } { c } { }
      \@@_Hdotsfor_i
    }
    { \@@_fatal:n { Hdotsfor-in-row-0 } }
  }

  \multicolumn { \l_@@_first_row_int } { c } { }
  \@@_Hdotsfor_i
\}
\}
\end{verbatim}

For the row, it’s easy.

\begin{verbatim}
\int_set:Nn \l_@@_initial_i_int { #1 }
\int_compare:nNnT \l_@@_initial_i_int \l_@@_first_i_int
\{
\}
\end{verbatim}
We declare all the cells concerned by the \Hdotsfor as “dotted” (for the dotted lines created by \Cdots, \Ldots, etc., this job is done by \@@_find_extremities_of_line:nnnn). This declaration is done by defining a special control sequence (to nil).

\int_step_inline:nnn { #2 } { #2 + #3 - 1 } { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }

\hook_gput_code:nnn { begindocument } { . }

\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
For the column, it's easy.

\int_set:Nn \l_@@_initial_j_int { #2 }
\int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

For the row, it's a bit more complicated.

\int_compare:nNnTF { #1 } = 1
{ \int_set:Nn \l_@@_initial_i_int 1
  \bool_set_true:N \l_@@_initial_open_bool
}
{ \cs_if_exist:cTF { pgf @ sh @ ns @ \@@_env:
    - \int_eval:n { #1 - 1 }
    - \int_use:N \l_@@_initial_j_int
    } \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }\int_set:Nn \l_@@_initial_i_int { #1 }
\bool_set_true:N \l_@@_initial_open_bool
\int_compare:nNnTF { #2 } = 0
{ \color { nicematrix-first-col } }
{ \int_compare:nNnT { #2 } = \g_@@_col_total_int
  { \color { nicematrix-last-col } } }
We declare all the cells concerned by the \texttt{\Vdotsfor} as “dotted” \cite{ref:1} (for the dotted lines created by \texttt{\Cdots}, \texttt{\Ldots}, etc., this job is done by \texttt{\@@\_find\_extremities\_of\_line:nnnn}). 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 \texttt{\@@rotate:} will be linked to \texttt{\rotate} in \texttt{\{NiceArrayWithDelims\}}.

\begin{verbatim}
\cs_new_protected:Npn \@@rotate: { \bool_gset_true:N \g_@@_rotate_bool }
\end{verbatim}

\textbf{The command \texttt{\line} accessible in code-after}

In the \texttt{\CodeAfter}, the command \texttt{\@@\_line:nnnn} 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 the following behaviour:

- If the argument is of the format \texttt{i-j}, our command applies the command \texttt{\int\_eval:n} to \texttt{i} and \texttt{j};
- If not (that is to say, when it's a name of a \texttt{\Block}), the argument is left unchanged.

This must \textit{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
{ \tl_if_empty:nTF { #2 } { #1 } { \@@\_double\_int\_eval_i:n #1-#2- \q_stop } }
\cs_new:Npn \@@\_double\_int\_eval_i: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:nnnn} (the construction uses the fact the \texttt{\@@\_line:nnnn} is protected and that \texttt{\@@\_double\_int\_eval:n} is fully expandable).

\begin{verbatim}
\hook_gput_code:nnn { \begindocument } { . }
\keys_set:nn { \NiceMatrix / \xdots } { #4 }
\tl_if_empty:VF \l_\@@_xdots\_color\_tl { \color { \l_\@@_xdots\_color\_tl } }
\@@\_actually\_draw\_Vdots:
\group_end:
\end{verbatim}
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:\n
\c_@@_pgfortikzpicture_tl
\@@_draw_line_iii:nn { #1 } { #2 }
\c_@@_endpgfortikzpicture_tl

The following command must be protected (it’s used in the construction of \@@_draw_line_ii:nn).

\keys_define:nn { NiceMatrix / RowStyle }
{ cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
cell-space-top-limit .initial:n = \c_zero_dim ,
cell-space-top-limit .value_required:n = true ,
cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
cell-space-bottom-limit .initial:n = \c_zero_dim ,
cell-space-bottom-limit .value_required:n = true ,
cell-space-limits .meta:n =
  { cell-space-top-limit = #1 ,
    cell-space-bottom-limit = #1 ,
  },
color .tl_set:N = \l_tmpa_tl ,
color .value_required:n = true ,
bold .bool_set:N = \l_tmpa_bool ,
bold .default:n = true ,
bold .initial:n = false ,
nb-rows .code:n =
  \str_if_eq:nnTF { #1 } { * }
  { \int_set_eq:NN \l_@@_key_nb_rows_int 500 }
  { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
nb-rows .value_required:n = true ,
rowcolor .tl_set:N = \l_@@_tmpc_tl ,
rowcolor .value_required:n = true ,
rowcolor .initial:n = ,
unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
\NewDocumentCommand \@@_RowStyle:n { O { } m }
  { \tl_clear:N \l_tmpa_tl
    \int_set:Nn \l_@@_key_nb_rows_int 1
    \keys_set:nn { NiceMatrix / RowStyle } { #1 }
    If the key rowcolor has been used.
    \tl_if_empty:NF \l_@@_tmpc_tl 
    First, the end of the current row (we remind that \RowStyle applies to the end of the current row).
    \tl_gput_right:Nx \g_nicematrix_code_before_tl
    \{ \@@_rectanglecolor
      \{ \l_@@_tmpc_tl \}
      \{ \int_use:N \c@iRow - \int_use:N \c@jCol \}
      \{ \int_use:N \c@iRow - * \}
    \}

    Then, the other rows (if there is several rows).
    \int_compare:nNnT \l_@@_key_nb_rows_int > 1
    \{ \tl_gput_right:Nx \g_nicematrix_code_before_tl
      \{ \@@_rowcolor
        \{ \l_@@_tmpc_tl \}
        \{ \int_eval:n { \c@iRow + 1 } - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 } \}
      \}
    \}
    \}
    \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
    \tl_gput_right:Nn \g_@@_row_style_tl { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
    \tl_gput_right:Nn \g_@@_row_style_tl { \#2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
\dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
\{ \tl_gput_right:Nx \g_@@_row_style_tl
  \{ \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
\}
\dim_set:Nn \l_@@_cell_space_top_limit_dim
{ \dim_use:N \l_tmpa_dim }
\dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
{ \tl_gput_right:Nx \g_@@_row_style_tl
  \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
  \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
  { \dim_use:N \l_tmpb_dim }
}
\l_tmpa_tl is the value of the key \texttt{color} of \RowStyle.
\tl_if_empty:NF \l_tmpa_tl
{ \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
\l_tmpa_bool is the value of the key \texttt{bold}.
\bool_if:NT \l_tmpa_bool
{ \if_mode_math:
  \c_math_toggle_token
  \bfseries \boldmath
  \c_math_toggle_token
  \else:
  \bfseries \boldmath
  \fi:
}
\tl_gput_right:Nn \g_@@_row_style_tl { \fi }
\g_@@_row_style_tl \ignorespaces
\)

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction \pgfusepath { fill } (and they will be in the same instruction \texttt{fill}—coded \texttt{f}—in the resulting PDF).

The commands \@@_rowcolor, \@@_columncolor, \@@_rectanglecolor and \@@_rowlistcolors don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence \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 (eg: \texttt{[gray]{0.5}}).

- For the color whose index in \g_@@_colors_seq is equal to \texttt{i}, a list of instructions which use that color will be constructed in the token list \g_@@_color_i_tl. In that token list, the instructions will be written using \@@_cartesian_color:nn and \@@_rectanglecolor:nn.

#1 is the color and #2 is an instruction using that color. Despite its name, the command \@@_add_to_colors_seq:nn doesn’t only add a color to \g_@@_colors_seq: it also updates the
We add in a global way because the final user may use the instructions such as \texttt{cellcolor} in a loop of \texttt{pgffor} in the \texttt{CodeBefore} (and we recall that a loop of \texttt{pgffor} is encapsulated in a group).

First, we look for the number of the color and, if it’s found, we store it in $\l_\text{tmpa\_int}$. If the color is not present in $\l_\text{@@\_colors\_seq}$, $\l_\text{tmpa\_int}$ will remain equal to 0.

First, the case where the color is a new color (not in the sequence).

Now, the case where the color is not a new color (the color is in the sequence at the position $\l_\text{tmpa\_int}$).

The macro \texttt{\@@\_actually\_color:} will actually fill all the rectangles, color by color (using the sequence $\l_\text{@@\_colors\_seq}$ and all the token lists of the form $\l_\text{@@\_color\_i\_tl}$).

Here is an example: \texttt{\@@\_rowcolor \{red!15\} \{1,3,5-7,10-\}}
Here an example:
\begin{verbatim}
\@@_columncolor:nn \{red!15\} \{1,3,5-7,10-\}
\end{verbatim}
\begin{verbatim}
\NewDocumentCommand \@@_columncolor { O { } m m }
\{ \tl_if_blank:nF { #2 } \{ \@@_add_to_colors_seq:xn \{ \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } \} \{ \@@_cartesian_color:nn { - } { #3 } \} \}
\}
\end{verbatim}

Here is an example:
\begin{verbatim}
\@@_rectanglecolor\{red!15\}\{2-3\}\{5-6\}
\end{verbatim}
\begin{verbatim}
\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 } \} \}
\}
\end{verbatim}

The last argument is the radius of the corners of the rectangle.
\begin{verbatim}
\NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
\{ \tl_if_blank:nF { #2 } \{ \@@_add_to_colors_seq:xn \{ \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } \} \{ \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } \} \}
\}
\end{verbatim}

The last argument is the radius of the corners of the rectangle.
\begin{verbatim}
\cs_new_protected:Npn \@@_rectanglecolor:nnn \l_@@_rows_tl \l_@@_cols_tl \l_@@_radius_tl
\{ \@@_cut_on_hyphen:w \l_@@_rows_tl \q_stop \tl_clear_new:N \l_@@_tmpc_tl \tl_clear_new:N \l_@@_tmpd_tl \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl \@@_cut_on_hyphen:w \l_@@_cols_tl \q_stop \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl } \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl } \}
\end{verbatim}

The command \@@_cartesian_path:n takes two implicit arguments: \l_@@_cols_tl and \l_@@_rows_tl.
\begin{verbatim}
\@@_cartesian_path:n { #3 }
\end{verbatim}

Here is an example:
\begin{verbatim}
\@@_cellcolor[rgb]\{0.5,0.5,0\}\{2-3,3-4,4-5,5-6\}
\end{verbatim}
\begin{verbatim}
\NewDocumentCommand \@@_cellcolor { O { } m m }
\{ \clist_map_inline:nn { #3 } { \@@_rectanglecolor \[ #1 \] { #2 } { ##1 } { ##1 } } \}
\end{verbatim}

\begin{verbatim}
\NewDocumentCommand \@@_chessboardcolors { O { } m m }
\{ \int_step_inline:nn { \int_use:N \c@iRow } \}
\end{verbatim}
The command \ @@_arraycolor (linked to \arraycolor at the beginning of the \CodeBefore) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the "corners".

\NewDocumentCommand \ @@_arraycolor { O { } m }
\{ \@@_rectanglecolor [ #1 ] { #2 }
\{ \int_use:N \c@iRow - \int_use:N \c@jCol \}
\}

\keys_define:nn { NiceMatrix / rowcolors }
\{ respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
respect-blocks .default:n = true ,
cols .tl_set:N = \l_@@_cols_tl ,
restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
restart .default:n = true ,
unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
\}

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 list of colors ; #4 is for the optional list of pairs key=value.

\NewDocumentCommand \ @@_rowlistcolors { O { } m m O { } }
\group_begin:
\seq_clear_new:N \l_@@_colors_seq
\seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
\tl_clear_new:N \l_@@_cols_tl
\tl_set:Nn \l_@@_cols_tl { - }
\keys_set:nn { NiceMatrix / rowcolors } { #4 }
\group_end:

The counter \l_@@_color_int will be the rank of the current color in the list of colors (modulo the length of the list).
\int_zero_new:N \l_@@_color_int
\int_set:Nn \l_@@_color_int 1
\bool_if:NF \l_@@_respect_blocks_bool
\{ \}

We don’t want to take into account a block which is completely in the “first column” of (number 0) or in the “last column” and that’s why we filter the sequence of the blocks (in a the sequence \l_tmpa_seq).
\seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
\seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq { \@@_not_in_exterior_p:nnnnn ##1 }

}
#2 is the list of intervals of rows.

\clist_map_inline:nn { #2 }
{
\tl_set:Nn \l_tmpa_tl { ##1 }
\tl_if_in:NnTF \l_tmpa_tl { - }
{ \@@_cut_on_hyphen:w ##1 \q_stop }
{ \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
\}

Now, \l_tmpa_tl and \l_tmpb_tl are the first row and the last row of the interval of rows that we have to treat. The counter \l_tmpa_int will be the index of the loop over the rows.

\int_set:Nn \l_tmpa_int \l_tmpa_tl
\bool_if:NTF \l_@@_rowcolors_restart_bool
{ \int_set:Nn \l_@@_color_int 1 }
{ \int_set:Nn \l_@@_color_int \l_tmpa_tl }
\int_zero_new:N \l_@@_tmpc_int
\int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
\int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
{
\tl_set:Nx \l_@@_rows_tl { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
\tl_clear_new:N \l_@@_color_tl
\tl_set:Nx \l_@@_color_tl { \@@_color_index:n { \int_mod:nn { \l_@@_color_int - 1 } { \seq_count:N \l_@@_colors_seq } + 1 } }
\int_if_empty:NF \l_@@_color_tl
{ \@@_add_to_colors_seq:xx { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } } \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
\int_incr:N \l_@@_color_int
\int_set:Nn \l_@@_color_int \l_tmpa_int + 1
}
\endpgfpicture
\group_end:

The command \@@_color_index:n peeks in \l_@@_colors_seq the color at the index #1. However, if that color is the symbol =, the previous one is poken. This macro is recursive.
\cs_new:Npn \@@_color_index:n { #1 }
The command \rowcolors (available in the \CodeBefore) is a specialisation of the most general command \rowlistcolors.

\NewDocumentCommand \@@_rowcolors { O { } m m m O { } } { \@@_rowlistcolors \[ #1 \] { #2 } { { #3 } , { #4 } } \[ #5 \] }

\cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
{ \int_compare:nNnT { #3 } > \l_tmpb_int
{ \int_set:Nn \l_tmpb_int { #3 } }
}

\prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
{ \bool_lazy_or:nnTF
{ \int_compare_p:nNn { #4 } = \c_zero_int }
{ \int_compare_p:nNn { #2 } = \{ \int_eval:n \{ \c@jCol + 1 \} } }
\prg_return_false:
\prg_return_true:
}

The following command return true when the block intersects the row \l_tmpa_int.

\prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
{ \bool_if:nTF
{ \int_compare:n { #1 <= \l_tmpa_int } 
&& \int_compare:n { \l_tmpa_int <= #3 } }
\prg_return_true:
\prg_return_false:
}

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 define below a command \@@_cartesian_path: which corresponds to a value 0 pt for the radius of the corners. This command is in particular used in \@@_rectanglecolor:nnn (used in \@@_rectanglecolor, itself used in \@@_cellcolor).

\cs_new_protected:Npn \@@_cartesian_path:n #1
{ \bool_lazy_and:nnT
{ ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
{ \dim_compare_p:nNn { #1 } = \c_zero_dim }
{ \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
\@@_expand_clist:NN \l_@@_rows_tl \c@iRow
}
\clist_map_inline:Nn \l_@@_cols_tl
{ \tl_set:Nn \l_tmpa_tl { ##1 }
\tl_if_in:NnTF \l_tmpa_tl { - }
}

We begin the loop over the columns.
\clist_map_inline:Nn \l_@@_cols_tl
{ \tl_set:Nn \l_tmpa_tl { ##1 }
\tl_if_in:NnTF \l_tmpa_tl { - }
\l_{\@\@\_tmpc\_tl} will contain the number of columns.

If we decide to provide the commands $\cellcolor$, $\rectanglecolor$, $\rowcolor$, $\columncolor$, $\rowcolors$ and $\chessboardcolors$ in the code-before of a $\SubMatrix$, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

\begin{verbatim}
\@@_qpoint:n \{ col - \l_{\@\@\_tmpa\_tl} \}
\int_compare:nNnTF \l_{\@\@\_first\_col\_int} = \l_{\@\@\_tmpa\_tl} 
{ \dim_set:Nn \l_{\@\@\_tmpc\_dim} { \pgf@x - 0.5 \arrayrulewidth } } 
{ \dim_set:Nn \l_{\@\@\_tmpc\_dim} { \pgf@x + 0.5 \arrayrulewidth } } 
\@@_qpoint:n \{ col - \int_eval:n \{ \l_{\@\@\_tmpb\_tl} + 1 \} \}
\dim_set:Nn \l_{\@\@\_tmpa\_dim} { \pgf@x + 0.5 \arrayrulewidth }
\end{verbatim}

We begin the loop over the rows.

\begin{verbatim}
\clist_map_inline:Nn \l_{\@\@\_rows\_tl} \l_{\@\@\_tmpc\_tl}
{ \l_{\@\@\_set:Nn} \l_{\@\@\_tmpa\_tl} \{ ####1 \}
 \l_{\@\@\_if_in:NnTF} \l_{\@\@\_tmpa\_tl} \{ - \}
 \{ \@@_cut_on_hyphen:w \####1 \q_stop \}
 \{ \@@_cut_on_hyphen:w \####1 - \####1 \q_stop \}
 \tl_if_empty:NT \l_{\@\@\_tmpa\_tl} { \tl_set:Nn \l_{\@\@\_tmpa\_tl} { 1 } }
 \tl_if_empty:NT \l_{\@\@\_tmpb\_tl}
{ \tl_set:Nx \l_{\@\@\_tmpb\_tl} { \int_use:N \c@iRow } }
\int_compare:nNnT \l_{\@\@\_tmpb\_tl} > \c@iRow 
{ \tl_set:Nx \l_{\@\@\_tmpb\_tl} { \int_use:N \c@iRow } }
\seq_if_in:NxF \l_{\@\@\_corners\_cells\_seq} \l_{\@\@\_corners\_cells\_seq} 
{ \l_{\@\@\_dim_set:Nn} \l_{\@\@\_tmpb\_dim} \{ \pgf@y + 0.5 \arrayrulewidth \}
 \l_{\@\@\_qpoint:n} \{ \int_eval:n \{ \l_{\@\@\_tmpb\_tl} + 1 \} \}
 \l_{\@\@\_dim_set:Nn} \l_{\@\@\_tmpb\_dim} \{ \pgf@y + 0.5 \arrayrulewidth \}
 \pgfsetcornersarced { \pgfpoint \l_{\@\@\_tmpc\_dim} \l_{\@\@\_tmpd\dim} }
 \pgfpathrectanglecorners 
{ \pgfpoint \l_{\@\@\_tmpb\_dim} \l_{\@\@\_tmpd\dim} }
 \pgfpoint \l_{\@\@\_tmpc\_dim} \l_{\@\@\_tmpd\dim} }
\end{verbatim}

Now, the numbers of both rows are in $\l_{\@\@\_tmpa\_tl}$ and $\l_{\@\@\_tmpb\_tl}$.

\begin{verbatim}
\seq_if_in:NxF \l_{\@\@\_corners\_cells\_seq} \l_{\@\@\_corners\_cells\_seq} 
{ \l_{\@\@\_dim_set:Nn} \l_{\@\@\_tmpa\_dim} \{ \pgf@x - 0.5 \arrayrulewidth \}
 \l_{\@\@\_qpoint:n} \{ \int_eval:n \{ \l_{\@\@\_tmpa\_tl} + 1 \} \}
 \l_{\@\@\_dim_set:Nn} \l_{\@\@\_tmpd\dim} \{ \pgf@x - 0.5 \arrayrulewidth \}
 \pgfsetcornersarced { \pgfpoint \l_{\@\@\_tmpc\_dim} \l_{\@\@\_tmpd\dim} }
 \pgfpoint \l_{\@\@\_tmpa\_dim} \l_{\@\@\_tmpd\dim} }
\end{verbatim}

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands $\@\@\_rowcolors$, $\@\@\_columncolor$ and $\@\@\_rowcolor:n$ (used in $\@\@\_rowcolor$).

\begin{verbatim}
\cs_new_protected:Npn \@@\@\_cartesian_path: { \@@\@\_cartesian_path:n \{ 0 pt \} }
\end{verbatim}

The following command will be used only with $\l_{\@\@\_cols\_tl}$ and $\c@jCol$ (first case) or with $\l_{\@\@\_rows\_tl}$ and $\c@iRow$ (second case). For instance, with $\l_{\@\@\_cols\_tl}$ equal to 2,4,6,8-* and $\c@jCol$ equal to 10, the clist $\l_{\@\@\_cols\_tl}$ will be replaced by 2,4,5,6,8,9,10.
When the user uses the key colortbl-like, the following command will be linked to `\cellcolor` in the tabular.

```latex
\NewDocumentCommand \@@_cellcolor_tabular { O { } m } {
  \peek_remove_spaces:n {
    \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 `!` which may be activated by some packages (ex.: `babel` with the option `french` on latex and pdflatex).

```latex
\cellcolor [ #1 ] { \exp_not:n { #2 } } {
  \int_use:N \c@iRow - \int_use:N \c@jCol }
```

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

```latex
\NewDocumentCommand \@@_rowcolor_tabular { O { } m } {
  \peek_remove_spaces:n {
    \tl_gput_right:Nx \g_nicematrix_code_before_tl {
      \@@_rectanglecolor \[ #1 \] { \exp_not:n { #2 } } {
        \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
    }
  }
}
```

```latex
\rowcolor \[ #1 \] { \exp_not:n { #2 } } {
  \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol }
}
```

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

```latex
\NewDocumentCommand \@@_columncolor_preamble { O { } m } {
  \peek_remove_spaces:n {
    \tl_gput_right:Nx \g_nicematrix_code_before_tl {
      \@@_columncolor_preamble \[ #1 \] { \exp_not:n { #2 } } {
        \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol }
    }
  }
}
```

```latex
\columncolor_preamble \[ #1 \] { \exp_not:n { #2 } } {
  \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol }
}
```
With the following line, we test whether the cell is the first one we encounter in its column (don’t forget that some rows may be incomplete).

```latex
\int_compare:nNnT \c@jCol > \g@@col_total_int
```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```latex
\tl_gput_left:Nx \g_nicematrix_code_before_tl
\exp_not:N \columncolor [ #1 ]
\exp_not:n { #2 } \int_use:N \c@jCol
```

The vertical and horizontal rules

**OnlyMainNiceMatrix**

We give to the user the possibility to define new types of columns (with `\newcolumntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential external rows of the array.

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

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

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

```latex
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
\int_compare:nNnTF \l@@first_col_int = 0
\@@_OnlyMainNiceMatrix_i:n { #1 }
\int_compare:nNnTF \c@jCol = 0
\int_compare:nNnF \c@iRow = { -1 }
\int_compare:nNnF \c@iRow = \l@@last_row_int - 1 { #1 }
\{ \@@_OnlyMainNiceMatrix_i:n { #1 } \}
```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, after a potential vertical rule on the left side of the first cell. The command `@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must not be protected.

```latex
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
\int_compare:nNnTF \c@iRow = 0
\int_compare:nNnF \c@iRow = \l@@last_row_int { #1 }
\{ \int_compare:nNnF \c@iRow = \l@@last_row_int { #1 } \}
```

Remember that `\c@iRow` is not always inferior to `\l@@last_row_int` because `\l@@last_row_int` may be equal to −2 or −1 (we can’t write `\int_compare:nNnT \c@iRow < \l@@last_row_int`).
General system for drawing rules

When a command, environment or “subsystem” of \nicematrix wants to draw a rule, it will write in the internal \CodeAfter a command \@@_vline:n or \@@_hline:n. Both commands take as argument a list of key=value pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

It’s possible that the rule won’t be drawn continuously from start to end because of the blocks (created with the command \Block), the virtual blocks (created by \Cdots, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by \@@_vline_ii: and \@@_hline_ii:. Those commands use the following set of keys.

The vertical rules

The following command will be executed in the internal \CodeAfter. The argument \#1 is a list of key=value pairs.

The group is for the options.

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of \|c|c|c| but only two columns used).
\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_@@_tmpc_tl { \int_eval:n \l_@@_position_int }
\int_step_variable:nnNn \l_@@_start_int \l_@@_end_int \l_tmpa_tl
\tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
\int_zero:N \l_@@_start_int
\bool_gset_true:N \g_tmpa_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq { \@@_test_vline_in_block:nnnnn ##1 }
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq { \@@_test_vline_in_block:nnnnn ##1 }
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq { \@@_test_vline_in_stroken_block:nnnn ##1 }
\clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
\bool_if:NTF \g_tmpa_bool { \int_compare:nNnT \l_@@_local_start_int = 0 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl } } { \int_compare:nNnT \l_@@_local_start_int > 0 { \int_set_eq:NN \l_@@_local_end_int \l@@_end_int \@@_vline_ii: { \int_zero:N \l_@@_local_start_int } } { \int_compare:nNnT \l_@@_local_start_int > 0 { \int_set_eq:NN \l_@@_local_end_int \l@@_end_int \@@_vline_ii: } { \int_set:Nn \l_@@_local_start_int \l@@_local_start_int } } { \int_compare:nNnT \l_@@_local_start_int > 0 { \int_set_eq:NN \l_@@_local_end_int \l@@_end_int \@@_vline_ii: } { \int_set:Nn \l_@@_local_start_int \l@@_local_start_int } } { \int_compare:nNnT \l_@@_local_start_int > 0 { \int_set_eq:NN \l_@@_local_end_int \l@@_end_int \@@_vline_ii: } { \int_set:Nn \l_@@_local_start_int \l@@_local_start_int } } }

\cs_new_protected:Npn \@@_test_in_corner_v:
{ \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } } { \bool_if:NnF \g_@@_corners_clist \@@_test_in_corner_v: { \seq_if_in:NxT \l_@@_corners_cells_seq { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } } { \bool_set_false:N \g_tmpa_bool } } } { \seq_if_in:NxT \l_@@_corners_cells_seq { \l_tmpa_tl - \l_tmpb_tl } } { \int_compare:nNnTF \l_@@_local_start_int = 0 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl } } { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl } }
First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.
\pgfusepath{fill}
\group_end:
\endpgfpicture

The following code is for the case of a dotted rule (with our system of rounded dots).
\cs_new_protected:Npn \@@_vline_iv:
\begin{tikzpicture}
\pgfsetlinewidth{1.1 \arrayrulewidth}
\pgfsetrectcap
\pgfusepathqstroke
\end{tikzpicture}

The following code is for the case when the user uses the key \texttt{tikz} (in the definition of a customized rule by using the key \texttt{custom-line}).
\cs_new_protected:Npn \@@_vline_v:
\begin{tikzpicture}
\pgfsetlinewidth{1.1 \arrayrulewidth}
\pgfsetrectcap
\pgfusepathqstroke
\end{tikzpicture}

The command \texttt{\@@\_draw_vlines} draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as \texttt{\textbackslash Cdots}) and in the corners (if the key \texttt{corners} is used).
\cs_new_protected:Npn \@@\_draw_vlines:
\begin{tikzpicture}
\pgfsetlinewidth{1.1 \arrayrulewidth}
\pgfsetrectcap
\pgfusepathqstroke
\end{tikzpicture}
The following command will be executed in the internal \CodeAfter. The argument #1 is a list of key=value pairs of the form \texttt{NiceMatrix/Rules}.

\begin{verbatim}
\cs_new_protected:Npn \@@_hline:n #1
\group_begin:
\int_zero_new:N \l_@@_end_int
\int_set_eq:NN \l_@@_end_int \c@jCol
\keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
\@@_hline_i:
\group_end:
\end{verbatim}

The group is for the options.

\begin{verbatim}
\cs_new_protected:Npn \@@_hline_i:
\int_zero_new:N \l_@@_local_start_int
\int_zero_new:N \l_@@_local_end_int
\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.

\l_tmpa_tl \l_@@_position_int \l_@@_start_int \l_@@_end_int
\l_tmpb_tl
\l_@@_local_start_int \l_@@_local_end_int
\l_@@_other_keys_tl
\l_@@_corners_clist
\g_@@_pos_of_blocks_seq
\g_@@_pos_of_xdots_seq
\g_@@_pos_of_stroken_blocks_seq
\bool_gset_true:N \g_tmpa_bool
\bool_gset_true:N \g_tmpb_bool
\seq_map_inline:Nn \g_@@_pos_of_blocks_seq
\seq_map_inline:Nn \g_@@_pos_of_xdots_seq
\seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
\clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
\bool_if:NTF \g_tmpa_bool
\int_compare:nNnT \l_@@_local_start_int = 0
We keep in memory that we have a rule to draw. \l_@@_local_start_int will be the starting row of the rule that we will have to draw.
\end{verbatim}

\begin{verbatim}
\end{verbatim}
First the case of a standard rule (without the keys dotted and tikz).

\cs_new_protected:Npn \@@_hline_iii: 
{ 
\pgfpicture 
\pgfrememberpicturepositiononpagetrue 
\pgf@relevantforpicturesizefalse 
\l_tmpa_tl = \c@iRow + 1 
\seq_if_in:NxT \l_@@_corners_cells_seq { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl } { \bool_set_false:N \g_tmpa_bool } 
\seq_if_in:NxT \l_@@_corners_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_@@_corners_cells_seq { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl } { \bool_set_false:N \g_tmpa_bool } } } \tl_if_empty:NTF \l_@@_tikz_rule_tl \@@_hline_iii: \@@_hline_v: 
} 

First the case of a standard rule (without the keys dotted and tikz).
The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets ($\hline$ doesn’t).

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

But, if the user uses \texttt{margin}, the dotted line extends to have the same width as a \texttt{hline}.  

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by 0.5 \l_\@_xdots_inter_dim is ad hoc for a better result.

\begin{tikzpicture}
\pgfpathmoveto{\pgfqpoint{0pt}{-0.05\rulewidth}}
\pgfpathlineto{\pgfqpoint{\pgf@x-0.05\rulewidth}{\pgf@y}}
\end{tikzpicture}

The following code is for the case when the user uses the key \texttt{tikz} (in the definition of a customized rule by using the key \texttt{custom-line}).
The command \ @@_draw_hlines: draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as \Cdots and in the corners (if the key corners is used)).

\cs_new_protected:Npn \ @@_draw_hlines: \\
{ \int_step_inline:nnn \\
\{ \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool } 1 2 \} \\\n\{ \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool } \\\n\{ \int_eval:n { \c@iRow + 1 } \c@iRow \} \\\n\} \\\n\{ \tl_if_eq:NnF \l_@@_hlines_clist { all } \\\n\{ \clist_if_in:NnT \l_@@_hlines_clist { ##1 } \\\n\{ \@@_hline:n \{ position = ##1 , total-width = \arrayrulewidth \} \} \\\n\} \\\n\ifnum 0 = `{ \fi \}

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_remove_spaces:n \\\n\{ \peek_meaning:NTF \Hline \\\n\{ \@@_Hline_ii:nn \{ #1 + 1 \} \} \\\n\{ \@@_Hline_iii:n \{ #1 \} \} \\\n\} \\\n\cs_set:Npn \ @@_Hline_ii:nn #1 #2 \{ \@@_Hline_i:n \{ #1 \} \}

Customized rules defined by the final user

The final user can define a customized rule by using the key custom-line in \NiceMatrixOptions. That key takes in as value a list of key=value pairs.

Among the keys available in that list, there is the key letter to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys \{NiceMatrix / ColumnTypes\}. That set of keys is used
to store the characteristics of those types of rules for convenience: the keys of that set of keys won’t
never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

\keys_define:nn { NiceMatrix / ColumnTypes } { }

The following command will create the customized rule (it is executed when the final user uses the
key `custom-line`, for example in `\NiceMatrixOptions`).

\cs_new_protected:Npn \@@_custom_line:n #1
\{ \str_clear_new:N \l_@@_command_str
\str_clear_new:N \l_@@_letter_str
\keys_set_known:nnN { NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl
\}
\keys_define:nn { NiceMatrix / custom-line }
\{ % here, we will use change in the future to use .str_set:N
letter .code:n = \str_set:Nn \l_@@_letter_str { #1 } ,
letter .value_required:n = true ,
% here, we will use change in the future to use .str_set:N
command .code:n = \str_set:Nn \l_@@_command_str { #1 } ,
command .value_required:n = true ,
\}
\cs_new_protected:Npn \@@_custom_line_i:n #1
\{ \bool_lazy_and:nnTF
\{ \str_if_empty_p:N \l_@@_letter_str \}
\{ \str_if_empty_p:N \l_@@_command_str \}
\{ \@@_error:n { No-letter-and-no-command } \}
\{ \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl \}
\}
\keys_define:nn { NiceMatrix / custom-line }
\{ % here, we will use change in the future to use .str_set:N
\bool_lazy_and:nnTF
\{ \str_if_empty_p:N \l_@@_letter_str \}
\{ \str_if_empty_p:N \l_@@_command_str \}
\{ \@@_error:n { No-letter-and-no-command } \}
\{ \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl \}
\}
\cs_new_protected:Npn \@@_custom_line_i:n #1
\{ % the following flags will be raised when the keys tikz, dotted and color are used (in the
custom-line).
\bool_set_false:N \l_@@_tikz_rule_bool
\bool_set_false:N \l_@@_dotted_rule_bool
\bool_set_false:N \l_@@_color_bool
\keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
\bool_if:NT \l_@@_tikz_rule_bool
\{ \@@_error:n { tikz~in~custom-line~without~tikz } \}
\bool_if:NT \l_@@_color_bool
\{ \@@_error:n { color~in~custom-line~with~tikz } \}
\bool_if:nT
\{ \int_compare_p:nNn \l_@@_multiplicity_int > 1
\&& \l_@@_dotted_rule_bool
\}
\{ \@@_error:n { key-multiplicity-with-dotted } \}
\str_if_empty:NF \l_@@_letter_str
\{ \int_compare:nTF \{ \str_count:N \l_@@_letter_str != 1 \}
\{ \@@_error:n { Several-letters } \}
\}
\cs_if_exist:NF \tikzpicture
\{ \@@_error:n { tikz-in-custom-line-without-tikz } \}
\bool_if:NT \l_@@_color_bool
\{ \@@_error:n { color-in-custom-line-with-tikz } \}
\}
The final user can, locally, redefine a letter of column type. That’s compatible with the use of \keys_define:nx: the definition is local and may overwrite a previous definition.

\keys_define:nx { NiceMatrix / ColumnTypes }

\str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
\str_const:Nn \c_@@_forbidden_letters_str { lcrpmb|()[]!@<> }

The previous command \@@_custom_line_i:n uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of custom-line) will also be used further with other sets of keys (for instance \{NiceMatrix/Rules\}). That’s why the following set of keys has some keys which are no-op.

\keys_define:nn { NiceMatrix / custom-line-bis }

The following keys will indicate whether the keys dotted, tikz and color are used in the use of a custom-line.

\bool_new:N \l_@@_dotted_rule_bool
\bool_new:N \l_@@_tikz_rule_bool
\bool_new:N \l_@@_color_bool

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key width is deprecated and has been replaced by the key total-width.

\keys_define:nn { NiceMatrix / custom-line-width }
dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
}

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the 'h' in the name). #1 is the whole set of keys to pass to \@@_line:n.
\cs_new_protected:Npn \@@_h_custom_line:n #1
{
  We use \cs_set:cpn and not \cs_new:cpn because we want a local definition. Moreover, the command must not be protected since it begins with \noalign.
  \cs_set:cpn { nicematrix - \l_@@_command_str }
  \noalign
  \cs_set:cpn { nicematrix - \l_@@_command_str }
  \noalign
    \@@_compute_rule_width:n { #1 }
    \skip_vertical:n { \l_@@_rule_width_dim }
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      \@@_hline:n
      \#1 ,
      \total-width = \dim_use:N \l_@@_rule_width_dim
    }
  \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
  \cs_generate_variant:Nn \@@_h_custom_line:nn { n V }
  \cs_new_protected:Npn \@@_compute_rule_width:n #1
  {
    \bool_set_false:N \l_@@_tikz_rule_bool
    \bool_set_false:N \l_@@_total_width_bool
    \bool_set_false:N \l_@@_dotted_rule_bool
    \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
    \bool_if:NF \l_@@_total_width_bool
      \bool_if:NTF \l_@@_dotted_rule_bool
        \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim }
      { \bool_if:NF \l_@@_tikz_rule_bool
          \dim_set:Nn \l_@@_rule_width_dim
            \arrayrulewidth * \l_@@_multiplicity_int
            + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
        }
  }
  \cs_new_protected:Npn \@@_v_custom_line:n #1
  { \@@_compute_rule_width:n { #1 }
    \tl_gput_right:Nx \g_@@_preamble_tl
      \exp_not:N ! \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim }
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      \@@_vline:n
  }

In the following line, the \dim_use:N is mandatory since we do an expansion.
\tl_gput_right:Nx \g_@@_preamble_tl
  \exp_not:N ! \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim }
\tl_gput_right:Nx \g_@@_internal_code_after_tl
  \@@_vline:n
}
The key hvlines

The following command tests whether the current position in the array (given by \_tmpa\_tl for the row and \_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 \_tmpa\_bool is set to false.

\cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
\bool_lazy_all:nT
\begin{verbatim}
{ \int_compare_p:nNn \_tmpa\_tl > { #1 } }
{ \int_compare_p:nNn \_tmpa\_tl < { \_tmpb\_tl + 1 } }
{ \int_compare_p:nNn \_tmpb\_tl > { \_tmpa\_tl - 1 } }
{ \int_compare_p:nNn \_tmpb\_tl < { \_tmpa\_tl + 1 } }
\end{verbatim}
\bool_gset_false:N \_tmpa\_bool

The same for vertical rules.

\cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
\bool_lazy_all:nT
\begin{verbatim}
{ \int_compare_p:nNn \_tmpa\_tl > { #1 - 1 } }
{ \int_compare_p:nNn \_tmpa\_tl < { #3 + 1 } }
{ \int_compare_p:nNn \_tmpb\_tl > { #2 } }
{ \int_compare_p:nNn \_tmpb\_tl < { #4 + 1 } }
\end{verbatim}
\bool_gset_false:N \_tmpa\_bool

\cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
\bool_lazy_all:nT
\begin{verbatim}
{ \int_compare_p:nNn \_tmpa\_tl > { #1 - 1 } }
{ \int_compare_p:nNn \_tmpa\_tl < { \_tmpb\_tl + 2 } }
{ \int_compare_p:nNn \_tmpb\_tl > { #2 - 1 } }
{ \int_compare_p:nNn \_tmpb\_tl < { #4 + 1 } }
\end{verbatim}
\bool_gset_false:N \_tmpa\_bool

\cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4 #5
\bool_lazy_all:nT
\begin{verbatim}
{ \int_compare_p:nNn \_tmpa\_tl > { #1 - 1 } }
{ \int_compare_p:nNn \_tmpa\_tl < { #3 + 2 } }
{ \int_compare_p:nNn \_tmpb\_tl > { #2 - 1 } }
{ \int_compare_p:nNn \_tmpb\_tl < { #4 + 2 } }
\end{verbatim}
\bool_gset_false:N \_tmpa\_bool

184
The key corners

When the key \texttt{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.

\begin{verbatim}
\cs_new_protected:Npn \@@_compute_corners:
{\seq_clear_new:N \l_@@_corners_cells_seq
 \clist_map_inline:Nn \l_@@_corners_clist
 {\str_case:nnF {##1}\
 {NW} {\@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol}
 {NE} {\@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1}
 {SW} {\@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 \c@jCol}
 {SE} {\@@_compute_a_corner:nnnnnn \c@iRow \c@jCol \{ -1 \} \{ -1 \} 1 1}
}{\@@_error:nn {bad-corner} {##1}}
}\
\seq_if_empty:NF \l_@@_corners_cells_seq
{\tl_gput_right:Nx \g_@@_aux_tl
 \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
 {\seq_use:Nnnn \l_@@_corners_cells_seq , , ,}
}
\end{verbatim}

Even if the user has used the key \texttt{corners} the list of cells in the corners may be empty.

\begin{verbatim}
\seq_if_empty:NF \l_@@_corners_cells_seq
{\
 \tl_gput_right:Nx \g_@@_aux_tl
 \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
 {\seq_use:Nnnn \l_@@_corners_cells_seq , , ,}
}
\end{verbatim}

“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_@@_corners_cells_seq.

The six arguments of \texttt{\@@_compute_a_corner:nnnnnn} are as follow:

\begin{itemize}
\item #1 and #2 are the number of row and column of the cell which is actually in the corner;
\item #3 and #4 are the steps in rows and the step in columns when moving from the corner;
\item #5 is the number of the final row when scanning the rows from the corner;
\item #6 is the number of the final column when scanning the columns from the corner.
\end{itemize}

\begin{verbatim}
\cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
{\tl_gput_right:Nx \g_@@_aux_tl
 {\seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
 {\seq_use:Nnnn \l_@@_corners_cells_seq , , ,}
}
\end{verbatim}

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 \texttt{\l_tmpa_bool} will be raised when a non-empty cell is found.

\begin{verbatim}
\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}
\end{verbatim}
Now, you determine the last empty cell in the row of number 1.

\int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
\bool_set_false:N \l_tmpa_bool
\int_zero_new:N \l_@@_last_empty_column_int
\int_step_inline:nnnn { #1 } { #3 } { #5 }
\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_column_int { #1 } } }

Now, we loop over the rows.
\int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
\bool_set_false:N \l_tmpa_bool
\int_zero_new:N \l_@@_last_empty_column_int
\int_step_inline:nnnn { #1 } { #3 } { #5 }
\bool_lazy_or:nnTF
 { \cs_if_exist_p:c
     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - #1 } }
 \l_tmpb_bool
 { \bool_set_true:N \l_tmpa_bool }
 { \bool_if:NF \l_tmpa_bool
     { \int_set:Nn \l_@@_last_empty_column_int { #1 } } }

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 }
\bool_lazy_or:nNN
 { \cs_if_exist_p:c
     { pgf @ sh @ ns @ \@@_env: - #1 - #1 } }
 \l_tmpb_bool
 { \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_@@_corners_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 \diagbox).

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

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

\cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
\begin{Verbatim}
\int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
{ \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
{ \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
{ \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
{ \bool_set_true:N \l_tmpb_bool }
}
}

\end{Verbatim}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

\bool_new:N \l_@@_block_auto_columns_width_bool

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

\keys_define:nn \{ NiceMatrix / NiceMatrixBlock \}
\begin{Verbatim}
\auto-columns-width .code:n =
{ \bool_set_true:N \l_@@_block_auto_columns_width_bool
\dim_gzero_new:N \g_@@_max_cell_width_dim
\bool_set_true:N \l_@@_auto_columns_width_bool
}
\end{Verbatim}

The environment \{NiceMatrixBlock\}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

\bool_new:N \l_@@_block_auto_columns_width_bool

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

\keys_define:nn \{ NiceMatrix / NiceMatrixBlock \}
\begin{Verbatim}
\auto-columns-width .code:n =
{ \bool_set_true:N \l_@@_block_auto_columns_width_bool
\dim_gzero_new:N \g_@@_max_cell_width_dim
\bool_set_true:N \l_@@_auto_columns_width_bool
}
\end{Verbatim}
At the end of the environment \{NiceMatrixBlock\}, we write in the main aux file instructions for the column width of all the environments of the block (that's why we have stored the number of the first environment of the block in the counter \texttt{\_\_\_first_env_block_int}).

\begin{verbatim}
{\ool_if:NT \l_@@_block_auto_columns_width_bool 
\iow_shipout:Nn \@mainaux \ExplSyntaxOn 
\iow_shipout:Nx \@mainaux 
\cs_gset:cpn \l_@@_max_cell_width \int_use:N \g_@@_NiceMatrixBlock_int 
{ \dim_eval:n { \g_@@_max_cell_width_dim + \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_generate_variant:Nn \dim_min:nn { v n } \cs_generate_variant:Nn \dim_max:nn { v n }
\end{verbatim}

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 and that construction uses the standard medium nodes).

\begin{verbatim}
\cs_new_protected:Npn \@@_create_extra_nodes:
{ \bool_if:nTF \l_@@_medium_nodes_bool 
\bool_if:NTF \l_@@_large_nodes_bool 
\@@_create_medium_and_large_nodes: 
\@@_create_medium_nodes: 
\} ; \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: ; }
\end{verbatim}

We have three macros of creation of nodes: \texttt{\_\_\_create_medium_nodes::, \_\_\_create_large_nodes:: and \_\_\_create_medium_and_large_nodes::.}

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command \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{\_\_\_row_i_min_dim} and \texttt{\_\_\_row_i_max_dim}. The dimension \texttt{\_\_\_row_i_min_dim} is the minimal \texttt{y}-value of all the cells of the row \texttt{i}. The dimension \texttt{\_\_\_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{\_\_\_column_j_min_dim} and \texttt{\_\_\_column_j_max_dim}. The dimension \texttt{\_\_\_column_j_min_dim} is the minimal \texttt{x}-value of all the cells of the column \texttt{j}. The dimension \texttt{\_\_\_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_max_dim}.

\begin{verbatim}
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
{ \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \l_@@_i: 
\end{verbatim}

188
We begin the two nested loops over the rows and the columns of the array.

\begin{verbatim}
\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j: 
\{ 
\dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
\dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
\dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
\dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
\}
\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i: 
\{ 
\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j: 
\{ 
\dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
\dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
\dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
\dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
\}
\end{verbatim}

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.

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

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

\begin{verbatim}
\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i: 
\{ 
\dim_compare:nNnT { \dim_use:c { l_@@_row_\@@_i: _min_dim } } = \c_max_dim 
\{ 
\@@_qpoint:n { row - \@@_i: - base }
\dim_set:cn { l_@@_row_\@@_i: _max_dim } \pgf@y
\dim_set:cn { l_@@_row_\@@_i: _min_dim } \pgf@y
\}
\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j: 
\end{verbatim}

189
Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

\cs_new_protected:Npn \@@_create_medium_nodes:
\beginpgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:
\@@_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:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { -large }
\@@_create_nodes:
\endpgfpicture

\cs_new_protected:Npn \@@_create_large_nodes:
\beginpgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:
\@@_computations_for_large_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:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { -large }
\@@_create_nodes:
\endpgfpicture

\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
\beginpgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:
\@@_create_nodes:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { -medium }
\@@_create_nodes:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { -large }
\@@_create_nodes:
\endpgfpicture

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:
\begin{verbatim}
\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:
\begin{verbatim}
\int_step_variable:nNn { \c@jCol - 1 } \@@_j:
\end{verbatim}
\end{verbatim}

Here, we have to use \dim_sub:cn because of the number 1 in the name.

\int_step_variable:nnNn { \c@iRow - 1 } \@@_i:
\int_step_variable:nnNn { \c@jCol - 1 } \@@_j:
\end{verbatim}

The command \@@_create_nodes: is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions \l_@@_row_i_min_dim, \l_@@_row_i_max_dim, \l_@@_column_j_min_dim and \l_@@_column_j_max_dim. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses \l_@@_suffix_tl (-medium or -large).

\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:
\end{verbatim}

We draw the rectangular node for the cell (\@@_i-\@@_j).
\begin{verbatim}
\pgf_rect_node:nnnn { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
\end{verbatim}
Now, we create the nodes for the cells of the \textbackslash multicolumn. We recall that we have stored in \texttt{\g@@multicolumn\texttt{cells_seq}} the list of the cells where a \texttt{\multicolumn{\textit{n}}{...}{...}} with \textit{n}\textgreater{}1 was issued and in \texttt{\g@@multicolumn\texttt{sizes_seq}} the correspondent values of \textit{n}.

The command \texttt{\g@@node\texttt{for\texttt{multicolumn}:\texttt{nn}} takes two arguments. The first is the position of the cell where the command \texttt{\multicolumn{\textit{n}}{...}{...}} was issued in the format \textit{i}-\textit{j} and the second is the value of \textit{n} (the length of the “multi-cell”).

The blocks

The code deals with the command \textbackslash Block. This command has no direct link with the environment \texttt{\textit{NiceMatrixBlock}}.

The options of the command \textbackslash 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.

\begin{verbatim}
\keys_define:nn { NiceMatrix / Block / FirstPass }
{ l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
  l .value_forbidden:n = true ,
  r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
  r .value_forbidden:n = true ,
\end{verbatim}
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 `>`. It’s mandatory to use an expandable command.

```latex
\NewExpandableDocumentCommand \@@_Block: { O { } m D < > { } +m }
\peek_remove_spaces:n
\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.

```latex
\cs_new:Npn \@@_Block_i #1-#2 \q_stop \{ \@@_Block_ii:nnnnn \}
\cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
\bool_lazy_or:nnTF
\{ \tl_if_blank:p:n { #2 } \}
\{ \@@_Block_i 1-1 \q_stop \}
\{ \@@_Block_i #2 \q_stop \}
\{ #1 \} { #3 } { #4 }
\}
```

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=value` pairs, `#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.

```latex
\cs_new:Npn \@@_Block_ii:nnnnn \}
\cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
\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_tmpb_int { #1 } \}
```

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).
If the block is mono-column.

\int_compare:nNnTF \l_\tmpb_int = 1
\str_if_empty:NTF \l_@@_hpos_cell_str
{ \str_set:Nn \l_@@_hpos_block_str c }
{ \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
\str_set:Nn \l_@@_hpos_block_str c

The value of \l_@@_hpos_block_str may be modified by the keys of the command \Block that we will analyze now.

\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, \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: \@@_Block_iv:nnnnn and \@@_Block_v:nnnnn (the five arguments of those macros are provided by curryfication).

\bool_if:nTF
{ \int_compare_p:nNn { \l_tmpa_int } = 1 || \int_compare_p:nNn { \l_tmpb_int } = 1 }
&& \tl_if_empty_p:n { #5 }
\exp_args:Nxx \@@_Block_iv:nnnnn
\exp_args:Nxx \@@_Block_v:nnnnn
\l_tmpa_int \l_tmpb_int { #3 } { #4 } { #5 }

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

\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 }
194

The following macro is for the case of a \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 PGF).

\cs_new_protected:Npn \@@_Block_iv:nnnnn \@@_Block_v: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 }
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 \set@color and not \color_ensure_current: (in order to use \color_ensure_current: safely, you should load \l3backend before the \documentclass with \RequirePackage{expl3}).

\tl_if_empty:NTF \l_@@_color_tl
\{ \int_compare:nNnT { #2 } = 1 \set@color \}
\{ \color { \l_@@_color_tl } \}

If the block is mono-row, we use \g_@@_row_style_tl even if it has yet been used in the beginning of the cell where the command \Block has been issued because we want to be able to take into account a potential instruction of color of the font in \g_@@_row_style_tl.

\int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
\group_begin:
\bool_if:NT \l_@@_respect_arraystretch_bool
{ \cs_set:Npn \arraystretch { 1 } }
\dim_zero:N \extrarowheight

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

\bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
\bool_if:NT \l_@@_NiceTabular_bool
{ \bool_lazy_all:nTF
\{ \int_compare_p:nNn { #2 } = 1 \}
\{ \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } \}
\{ ! \l_@@_respect_arraystretch_bool \}
\}

When the block is mono-column in a column with a fixed width (eg p{3cm}).

\begin { minipage } \[ \l_@@_vpos_of_block_tl \]
\l_@@_col_width_dim
\str_case:Vn \l_@@_hpos_block_str
\{ c \centering
\r \raggedleft
\l \raggedright
\}
#5
\end { minipage }
\}
\{ \use:x
\{ \exp_not:N \begin { tabular } \[ \l_@@_vpos_of_block_tl \]
\{ \@ { } \l_@@_hpos_block_str \@ { } \}
\}

If we are in a mono-column block, we take into account the width of that block for the width of the column.

\int_compare:nNnT { #2 } = 1
{\dim_set:Nn \g_@@_blocks_wd_dim
  { \dim_max:nn \g_@@_blocks_wd_dim
    { \box_wd:c { g_@@_block_box \int_use:N \g_@@_block_box_int_box } } }
}

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.

\int_compare:nNnT { #1 } = 1
{\dim_set:Nn \g_@@_blocks_ht_dim
  { \dim_max:nn \g_@@_blocks_ht_dim
    { \box_ht:c { g_@@_block_box \int_use:N \g_@@_block_box_int_box } } }
\dim_set:Nn \g_@@_blocks_dp_dim
  { \dim_max:nn \g_@@_blocks_dp_dim
    { \box_dp:c { g_@@_block_box \int_use:N \g_@@_block_box_int_box } } }
}
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_@@_hpos_block_str. However, maybe there were no key of the horizontal alignment and that’s why we put a key corresponding to the value of \l_@@_hpos_block_str, which is fixed by the type of current column.

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.

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).
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 \}
\{ tikz .code:n = \bool_if:NTF \c_@@_tikz_loaded_bool
\{ \seq_put_right:Nn \l_@@_tikz_seq \{ \#1 \} \}
\} \end \keys_define:nn
The command \@@_draw_blocks: will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of \ialign because there may be tabulars in the \Block instructions that will be composed now.

\cs_new_protected:Npn \@@_draw_blocks: 
\seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
\cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6 
\int_zero_new:N \l_@@_last_row_int 
\int_zero_new:N \l_@@_last_col_int 
\int_compare:nNnTF { #3 } > { 99 } 
{ \int_set_eq:NN \l_@@_last_row_int \c@iRow } 
{ \int_set:Nn \l_@@_last_row_int { #3 } } 
\int_compare:nNnTF { #4 } > { 99 } 
{ \int_set_eq:NN \l_@@_last_col_int \c@jCol } 
{ \int_set:Nn \l_@@_last_col_int { #4 } } 
\int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_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: 
{ \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } } 
\int_compare:nNnTF \l_@@_last_col_int <= \g_@@_static_num_of_col_int 
{ \l_@@_last_col_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: 
{ \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } } 
\int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int 
\msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } 
\@@_msg_redirect_name:nn { Block-too-large-1 } { none } 
\\Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } 
\\Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } 
\group_begin: 
\keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
We restrict the use of the key \texttt{v-center} to the case of a mono-row block.

\begin{verbatim}
\bool_if:NT \l_@@_v_center_bool
  \{ \int_compare:nNnF { #1 } = { #3 } \{
    \@@_error:n { Wrong~use~of~v-center } \bool_set_false:N \l_@@_v_center_bool
  \}
\}
\bool_if:NT \l_@@_vlines_block_bool
  \{ \tl_gput_right:Nx \g_nicematrix_code_after_tl
    \{ \exp_not:n { \#5 } \}
    \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int \}
\}
\bool_if:NT \l_@@_hlines_block_bool
  \{ \tl_gput_right:Nx \g_nicematrix_code_after_tl
    \{ \exp_not:n { \#5 } \}
    \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int \}
\}
\bool_if:nT \{ ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool \}
{ \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
  \{ \l_@@_block_name_str \}
}
\tl_if_empty:NF \l_@@_draw_tl
  \{ \tl_gput_right:Nx \g_nicematrix_code_after_tl
    \{ \exp_not:n { \#5 } \}
    \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int \}
\seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
  \{ \l_@@_block_name_str \}
\}
\clist_if_empty:NF \l_@@_borders_clist
  \{ \tl_gput_right:Nx \g_nicematrix_code_after_tl
    \{ \exp_not:n { \#5 } \}
    \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int \}
\}
\tl_if_empty:NF \l_@@_fill_tl
  \{
\end{verbatim}

The sequence of the positions of the blocks (excepted the blocks with the key \texttt{hvelines}) will be used when drawing the rules (in fact, there is also the \texttt{\multicolumn} and the \texttt{\diagbox} in that sequence).

\begin{verbatim}
\seq_gput_left:Nx \g_@@_pos_of_blocks_seq
  \{ \l_@@_block_name_str \}
}
\end{verbatim}
The command \@\_extract\_brackets will extract the potential specification of color space at the beginning of \l_@@\_fill\_tl and store it in \l_tmpa\_tl and store the color itself in \l_tmpb\_tl.

6186 \exp_last_unbraced:NV \@@\_extract\_brackets \l_@@\_fill\_tl \q_stop
6187 \tl_gput_right:Nx \g_nicematrix_code\_before\_tl
6188 \exp_not:N \roundedrectanglecolor
6189 \[ \l_tmpa\_tl \]
6190 \{ \exp_not:N \l_tmpb\_tl \}
6191 \{ #1 - #2 \}
6192 \{ \int_use:N \l_@@\_last\_row\_int - \int_use:N \l_@@\_last\_col\_int \}
6193 \{ \dim_use:N \l_@@\_rounded\_corners\_dim \}
6194 
6195 \}
6196 
6197 \seq_if_empty:NF \l_@@\_tikz\_seq
6198 {
6199 \tl_gput_right:Nx \g_nicematrix_code\_before\_tl
7200 \}
7201 \\@@\_block\_tikz:nnnnnn
7202 \{ #1 \}
7203 \{ #2 \}
7204 \{ \int_use:N \l_@@\_last\_row\_int \}
7205 \{ \int_use:N \l_@@\_last\_col\_int \}
7206 \{ \seq_use:Nn \l_@@\_tikz\_seq { , } \}
7207 
7208 
7209 \cs_set_protected_nopar:Npn \diagbox \#1 \#2
7210 {
7211 \tl_gput_right:Nx \g_@@\_internal\_code\_after\_tl
7212 
7213 \\@@\_actually\_diagbox:nnnnnn
7214 \{ #1 \}
7215 \{ #2 \}
7216 \{ \int_use:N \l_@@\_last\_row\_int \}
7217 \{ \int_use:N \l_@@\_last\_col\_int \}
7218 \{ \exp_not:n { \#1 } \} \{ \exp_not:n { \#2 } \}
7219 
7220 
7221 \hbox_set:Nn \l_@@\_cell\_box \{ \set@color \#6 \}
7222 \bool_if:NT \g_@@\_rotate_bool \@@\_rotate\_cell\_box:

Let’s consider the following \{Nice\Tabular\}. Because of the instruction !{\hspace{1cm}} in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create two nodes relative to the block: the node 1-1-block and the node 1-1-block-short.

\begin{Nice\Tabular}\{cc!{\hspace{1cm}}c\}
\Block{2-2}{our block} & & one \\\n & & two \\\nthree & four & five \\\nsix & seven & eight \\\n\end{Nice\Tabular}

We highlight the node 1-1-block We highlight the node 1-1-block-short

<table>
<thead>
<tr>
<th>our block</th>
<th>one</th>
</tr>
</thead>
<tbody>
<tr>
<td>two</td>
<td></td>
</tr>
<tr>
<td>three</td>
<td>four</td>
</tr>
<tr>
<td>four</td>
<td>five</td>
</tr>
<tr>
<td>six</td>
<td>seven</td>
</tr>
<tr>
<td>seven</td>
<td>eight</td>
</tr>
</tbody>
</table>

201
We construct the node for the block with the name (#1-#2-block).
The function \(\texttt{@@_pgf_rect_node:nnnnn}\) takes in as arguments the name of the node and the four
coordinates of two opposite corner points of the rectangle.

Now, we create the “short node” which, in general, will be used to put the label (that is to say the
content of the node). However, if one of the keys \(L\), \(C\), or \(R\) is used (that information is provided by the
boolean \(\texttt{l@@_hpos_of_block_cap_bool}\)), we don’t need to create that node since the normal node
is used to put the label.

The short node is constructed by taking into account the contents of the columns involved in at least
one cell of the block. That’s why we have to do a loop over the rows of the array.

We recall that, when a cell is empty, no (normal) node is created in that cell. That’s why we test the
existence of the node before using it.

If all the cells of the column were empty, \(\texttt{l_@@_tmpb_dim}\) has still the same value \(\texttt{c_max_dim}\). In that
case, you use for \(\texttt{l_@@_tmpb_dim}\) the value of the position of the vertical rule.
If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function \@@_pgf_rect_node:nnn takes in as arguments the name of the node and two PGF points.

\bool_if:NT \l_@@_medium_nodes_bool 
\{ 
\@@_pgf_rect_node:nnn 
\{ \@@_env: - #1 - #2 - block - medium \} 
\pgfpointanchor \{ \@@_env: - #1 - #2 - medium \} { north-west } 
\{ \@@_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 \texttt{\Block} of one row.

\bool_if:nTF \l_@@_medium_nodes_bool 
\{ 
\@@_pgf_rect_node:nnn 
\{ \@@_env: - #1 - #2 - block - medium \} 
\pgfpointanchor \{ \@@_env: - #1 - #2 - medium \} { north-west } 
\{ \@@_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_p:nNn \{ \#1 \} = \{ \#3 \} \&\& \! \l_@@_v_center_bool 
\{
\int_compare:nNnT \{ \#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 \l_tampa_dim.

\pgfextracty \l_tampa_dim \{ \@@_qpoint:n \{ row - #1 - base \} \}

We retrieve (in \texttt{\pgf@x}) the x-value of the center of the block.

\pgfpointanchor
We put the label of the block which has been composed in \l@@cell_box.

\pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
\pgfset { inner~sep = \c_zero_dim }
\pgfnode { rectangle }
\str_case:Vn \l@@hpos_block_str
{ c { center } l { west } r { east }
}
{ \box_use_drop:N \l@@cell_box } { } { }

If the number of rows is different of 1, we will put the label of the block by using the short node (the label of the block has been composed in \l@@cell_box).

\int_compare:nNnT { #2 } = 0
{ \str_set:Nn \l@@hpos_block_str r }
\bool_if:nT \g@@last_col_found_bool
{ \int_compare:nNnT { #2 } = \g@@col_total_int
{ \str_set:Nn \l@@hpos_block_str l }
}
\pgftransformshift
{ \pgfpointanchor
\@@env: - #1 - #2 - block
\bool_if:NF \l@@hpos_of_block_cap_bool { - short }
}
\str_case:Vn \l@@hpos_block_str
{ c { center } l { west } r { east }
}
\pgfset { inner~sep = \c_zero_dim }
\pgfnode { rectangle }
\str_case:Vn \l@@hpos_block_str
{ c { center } l { west }
}
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).

If the user has used the key color of the command \Block without value, the color fixed by \arrayrulecolor is used.
We can’t use \pgfusepathqstroke because of the key rounded-corners.

\pgfusepath { stroke }

\endpgfpicture
\group_end:

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 \@@_vlines_block:nnn is a list of options for the rules that we will draw. 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 \@@_vlines_block:nnn #1 #2 #3
{
\dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
\keys_set_known:n { NiceMatrix / BlockBorders } { #1 }
\@@_cut_on_hyphen:w #2 \q_stop
\tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
\tl_set_eq:NN \l_@@_tmpd_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 } }
\int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
{
    \use:x
    {
        \@@_vline:n
    {
        position = ##1 ,
        start = \l_@@_tmpc_tl ,
        end = \int_eval:n { \l_tmpa_tl - 1 }
    }
}
}
\cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
{
\dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
\keys_set_known:n { NiceMatrix / BlockBorders } { #1 }
\@@_cut_on_hyphen:w #2 \q_stop
\tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
\tl_set_eq:NN \l_@@_tmpd_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 } }
\int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
{
    \use:x
    {
        \@@_hline:n
    }
}
}
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 \textit{i-j}) and the third is the last cell of the block (with the same syntax).

\begin{verbatim}
\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 } }
 { \tl_clear_new:N \l_@@_borders_tikz_tl
 \keys_set:nV { NiceMatrix / OnlyForTikzInBorders } \l_@@_borders_clist
 \@@_cut_on_hyphen:w #2 \q_stop
 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
 \tl_set_eq:NN \l_@@_tmpd_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 } }
 \@@_stroke_borders_block_i:
 }
 \hook_gput_code:nnn { begindocument } { . }
 { \cs_new_protected:Npx \@@_stroke_borders_block_i:
 { \c_@@_pgfortikzpicture_tl \@@_stroke_borders_block_ii:
 \c_@@_endpgfortikzpicture_tl }
 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
 { \pgfrememberpicturepositiononpagetrue \pgf@relevantforpicturesizefalse \CT@arc@
 \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_@@_tmpc_tl }
 }
 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
 { tikz .code:n = \cs_if_exist:NTF \tikzpicture
 { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
 }
\end{verbatim}
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 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 { #1 } \tl_if_empty:NTF \l_@@_borders_tikz_tl {
\pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
\pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
\pgfusepathqstroke
} {
\use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
}\}
\}

The following command is used to stroke the top border and the bottom border. The argument \#1 is the number of row (in the sense of the row node).

\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 } \tl_if_empty:NTF \l_@@_borders_tikz_tl {
\pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
\pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
\pgfusepathqstroke
} {
\use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
}\}
\}

Here is the set of keys for the command \@@_strokeBorders_block:n.nn.

\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 , \}
The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments \#1 and \#2 are the coordinates of the first cell and \#3 and \#4 the coordinates of the last cell of the block. \#5 is a comma-separated list of the Tikz keys used with the path.

\cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
\begin{tikzpicture}
\clist_map_inline:nn { #5 }
\path [ ##1 ] ( #1 -| #2 ) rectangle ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
\end{tikzpicture}

How to draw the dotted lines transparently

\cs_set_protected:Npn \@@_renew_matrix:
\RenewDocumentEnvironment { pmatrix } { } { \pNiceMatrix }
\endpNiceMatrix
\RenewDocumentEnvironment { vmatrix } { } { \vNiceMatrix }
\endvNiceMatrix
\RenewDocumentEnvironment { Vmatrix } { } { \VNiceMatrix }
\endVNiceMatrix
\RenewDocumentEnvironment { bmatrix } { } { \bNiceMatrix }
\endbNiceMatrix
\RenewDocumentEnvironment { Bmatrix } { } { \BNiceMatrix }
\endBNiceMatrix

Automatic arrays

\cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
\int_set:Nn \l_@@_nb_rows_int { #1 }
\int_set:Nn \l_@@_nb_cols_int { #2 }
\keys_define:nn { NiceMatrix / Auto }
{l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,}
{r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,}
{c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c}
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
\int_zero_new:N \l_@@_nb_rows_int
\int_zero_new:N \l_@@_nb_cols_int
\keys_define:nn { NiceArrayWithDelims },
{l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,}
{r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,}
{c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c}
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
\int_zero_new:N \l_@@_nb_rows_int
\int_zero_new:N \l_@@_nb_cols_int
\keys_define:nn { NiceArrayWithDelims },
\begin{AutoNiceMatrixWithDelims}
\end{AutoNiceMatrixWithDelims}

The group is for the protection of `\l_@@_type_of_col_tl`.\group_begin:
We 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 a command \AutoNiceMatrix similar to the environment \{NiceMatrix\}.

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}`).

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of nicematrix. However, there are also redefinitions of `\diagbox` in other circumstances.

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `corners`.

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

The first four arguments of `\@@_actually_diagbox:nnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.
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.

\begin{minipage}{20 cm}
\raggedleft
\@math_toggle_token: #5 \@math_toggle_token:
\end{minipage}
\end{minipage}
\endpgfscope
\pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
\pgfnode { rectangle } { north-east }
{ }
{ }
\endpgfpicture

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 } { 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. 124.

In the environments of nicematrix, \CodeAfter will be linked to \@@_CodeAfter:. That macro must not be protected since it begins with \omit.

212
However, in each cell of the environment, the command \CodeAfter will be linked to the following command \@@_CodeAfter_ii:n which begins with \\.

We have to catch everything until the end of the current environment (of \nicematrix). First, we go until the next command \end.

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

If this is really the end of the current environment (of \nicematrix), we put back the command \end and its argument in the \TeX{} flow.

If this is not the \end we are looking for, we put those tokens in \g_nicematrix_code_after_tl and we go on searching for the next command \end with a recursive call to the command \@@_CodeAfter:n.

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 space between columns).

The first argument is the type of delimiter (\texttt{\)}, \texttt{\[}, \texttt{\{}, \texttt{\]} or \texttt{\}}). The second argument is the number of column. 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.

The \_\_\_y_initial_dim and \_\_\_y_final_dim will be the y-values of the extremities of the delimiter we will have to construct.

We will compute in \_\_\_tmpa_dim the x-value where we will have to put our delimiter (on the left side or on the right side).
Now we can put the delimiter with a node of PGF.
\pgfset { inner~sep = \c_zero_dim }
\dim_zero:N \nulldelimiterspace
\pgftransformshift
\pgfpointanchor \l_tmpa_dim { \bool_if:nTF { #3 } { west } { east } }
\dim_set:Nn \l_tmpa_dim { \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
\pgfnode { rectangle } { \bool_if:nTF { #3 } { east } { west } }
\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 { \nullfont \hrule \@height \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim } \@depth \c_zero_dim \@width \c_zero_dim }
\bool_if:nTF { #3 } { \right . } { \right #1 } \c_math_toggle_token
\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 .meta:n = { left-xshift = \#1, right-xshift = \#1 } ,
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 ,
\keys_define:nn { NiceMatrix }
{
  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 }
{
  delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
  delimiters / color .value_required: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 ,
  name .code:n = \tl_if_empty:nTF { #1 } { \@@_error:n { Invalid~name } }
{ \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #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 } }
},
  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 { } }
{
  \peek_remove_spaces:n
{ \@@_cut_on_hyphen:w #3 \q_stop
  \tl_clear_new:N \l_@@_tmpc_tl
  \tl_clear_new:N \l_@@_tmpd_tl
  \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
  \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
  \@@_cut_on_hyphen:w #2 \q_stop
  \seq_gput_right:Nx \g_@@_submatrix_seq

215
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;
- \#6 is the potential subscript;
- \#7 is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command \Cdots.

The following macro will compute \l_@@_first_i_tl, \l_@@_first_j_tl, \l_@@_last_i_tl and \l_@@_last_j_tl from the arguments of the command as provided by the user (for example 2-3 and 5-last).

\cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
\begin{verbatim}
{ \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 #1 \q_stop
  \tl_if_eq:NnTF \l_tmpa_tl { last } { \tl_set:NV \l_@@_first_i_tl \c@iRow }
  \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
  \tl_if_eq:NnTF \l_tmpb_tl { last } { \tl_set:NV \l_@@_first_j_tl \c@jCol }
  \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
  \@@_cut_on_hyphen:w #2 \q_stop
  \tl_if_eq:NnTF \l_tmpa_tl { last } { \tl_set:NV \l_@@_last_i_tl \c@iRow }
  \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
  \tl_if_eq:NnTF \l_tmpb_tl { last } { \tl_set:NV \l_@@_last_j_tl \c@jCol }
  \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
  \@@_cut_on_hyphen:w #1 \q_stop
  \tl_if_eq:NnTF \l_tmpa_tl { last } { \tl_set:NV \l_@@_first_i_tl \c@iRow }
  \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
  \tl_if_eq:NnTF \l_tmpb_tl { last } { \tl_set:NV \l_@@_first_j_tl \c@jCol }
  \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
  \@@_cut_on_hyphen:w #2 \q_stop
  \tl_if_eq:NnTF \l_tmpa_tl { last } { \tl_set:NV \l_@@_last_i_tl \c@iRow }
  \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
  \tl_if_eq:NnTF \l_tmpb_tl { last } { \tl_set:NV \l_@@_last_j_tl \c@jCol }
  \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
}
\end{verbatim}
The following token lists correspond to the position of the \SubMatrix.

\begin{verbatim}
\begin{array}{c}
\end{verbatim}
\end{array}

The last value of \texttt{\int_step_inline:nnn} is provided by currification.

\begin{verbatim}
\begin{array}{c}
\end{verbatim}
\end{array}

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

\begin{verbatim}
\begin{array}{c}
\end{verbatim}
\end{array}
We will draw the rules in the \SubMatrix.

%group_begin:
\pgfsetlinewidth { 1:1 \arrayrulewidth }
\tl_if_empty:NF \l_@@_rules_color_tl
{ \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
\CT@arc@

Now, we draw the 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.

\seq_map_inline:Nn \g_@@_cols_vlism_seq
{ \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
{ \int_compare:nNnT { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
{ \@@_qpoint:n { col - ##1 }
\pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
\pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
\pgfusepathqstroke
}
}

Now, we draw the vertical rules specified in the key \vlines of \SubMatrix. The last argument of \int_step_inline:nn or \clist_map_inline:Nn is given by curryfication.

\tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
{ \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
{ \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
\bool_lazy_and:nnTF
{ \int_compare_p:nNn { \l_@@_first_j_tl } > 0 }
{ \int_compare_p:nNn { \l_@@_last_j_tl } < { \l_@@_first_j_tl + 1 }
{ \@@_qpoint:n { col - \int_eval:n { \l_@@_first_j_tl + 1 } }
\pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
\pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
\pgfusepathqstroke
}
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.

\tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all } { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } } { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist } {
\bool_lazy_and:nnTF { \int_compare_p:nNn { ##1 } > 0 } { \int_compare_p:nNn { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } } {
\@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
\text{We use a group to protect } \l_tmpa_dim \text{ and } \l_tmpb_dim.
\group_begin:
\We compute in } \l_tmpa_dim \text{ 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 \text{ mm} } \}
\{ \dim_sub:Nn \l_tmpa_dim { 0.2 \text{ mm} } \}
\{ \dim_sub:Nn \l_tmpa_dim { 0.9 \text{ mm} } \}
\pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
\text{We compute in } \l_tmpb_dim \text{ 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 { #2 } {
\{ \dim_add:Nn \l_tmpb_dim { 0.9 \text{ mm} } \}
\{ \dim_add:Nn \l_tmpb_dim { 0.2 \text{ mm} } \}
\{ \dim_add:Nn \l_tmpb_dim { 0.9 \text{ 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:nnnn \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@ \text{ (the color of the rules).
Now, we deal with the left delimiter. Of course, the environment {pgfscope} is for the \pgftransformshift.
\begin { pgfscope }
\pgfusepathqstroke
\pgftransformshift
Now, we deal with the right delimiter.

\pgftransformshift
\pgfpoint
{ \l_{\text{\@_x_final_dim}} + \l_{\text{\@_submatrix_right_xshift_dim}} }
{ ( \l_{\text{\@_y_initial_dim}} + \l_{\text{\@_y_final_dim}} ) / 2 }
\str_if_empty:NTF \l_{\text{\@_submatrix_name_str}}
\{ \text{#0\_node\_right:nnn \#2 { } \#3 { } \#4 } \}
\{ \text{#0\_node\_right:nnn \#2} \}
\{ \text{#0\_env: - \l_{\text{\@_submatrix_name_str\_\_\_-right}}} \}
\end{pgfscope}

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}-\textit{j}, \textit{row}-\textit{i}, \textit{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.

\cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor
\flag_clear_new:n \l_{\text{\@_code_tl}}
\}

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 \#1 of \pgfpointanchor and we apply to it the command \@@_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.

\cs_new_protected:Npn \@@_pgfpointanchor:n \#1 \#2
{ \#1 { \@@_pgfpointanchor_ii:w \#2 - \q_stop } }

In fact, the argument of \pgfpointanchor is always of the form \texttt{\textbackslash a\_command \{ name\_of\_node \}} where “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 -.

\cs_new:Npn \@@_pgfpointanchor_i:nn \#1 \#2
{ \#1 { \@@_pgfpointanchor_ii:w \#2 - \q_stop } }

Since \seq_if_in:NnTF and \clist_if_in:NnTF are not expandable, we will use the following token list and \str_case:nVTF to test whether we have an integer or not.

\tl_const:Nn \l_{\text{\@_0\_integers\_alist_tl}}
{ { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { } }
{ { 1 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { } }
{ { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { } }
{ { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { } }
If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form \textit{i-j}. In that case, the \textit{i} of the number of row arrives first (and alone) in a \texttt{pgfpointanchor} and, the, the \textit{j} arrives (alone) in the following \texttt{pgfpointanchor}. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called \texttt{nicematrix}.

If there is an hyphen, we have to see whether we have a node of the form \texttt{i-j}, \texttt{row-i} or \texttt{col-j}.

Now the case of a node of the form \texttt{i-j}.

The command \texttt{\@node_left:nn} puts the left delimiter with the correct size. The argument \texttt{#1} is the delimiter to put. The argument \texttt{#2} is the name we will give to this PGF node (if the key \texttt{name} has been used in \texttt{\SubMatrix}).
\right. \c_math_toggle_token
\{ #2 \}
\}
\}

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). The argument #3 is the subscript and #4 is the superscript.

\cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
{\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\tl_if_empty:nF{#3}{_ {\smash{#3}}}^{\smash{#4}}\c_math_toggle_token}{#2}{}}

Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

\NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }{\peek_remove_spaces:n{\@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }}}
\NewDocumentCommand \@@_OverBrace { O { } m m m O { } }{\peek_remove_spaces:n{\@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }}}
\keys_define:nn { NiceMatrix / Brace }{left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,left-shorten .default:n = true ,right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,shorten .meta:n = { left-shorten , right-shorten },right-shorten .default:n = true ,yshift .dim_set:N = \l_@@_brace_yshift_dim ,yshift .value_required:n = true ,}
\begin{code}
\def\@@_brace:nnnnn#1#2#3#4#5{
    \group_begin:
    \edef\l_tmpa_tl{#1}
    \edef\l_tmpb_tl{#2}
    \edef\l_tmpc_tl{#3}
    \edef\l_tmpd_tl{#4}
    \edef\l_tmpe_tl{#5}
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \bool_if:NT \l_@@_brace_left_shorten_bool{
        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
        \int_step_inline:nnn {\l_@@_first_i_tl}{\l_@@_last_i_tl}{
            \cs_if_exist:cT{pgf @ sh @ ns @\@@_env: - ##1 - \l_@@_first_j_tl}{
                \pgfpointanchor{\@@_env: - ##1 - \l_@@_first_j_tl}{west}
                \dim_set:Nn \l_@@_x_initial_dim{\dim_min:nn \l_@@_x_initial_dim \pgf@x}
            }
        }
    }
    \bool_lazy_or:nnT{\bool_not_p:n \l_@@_brace_right_shorten_bool}{
        \dim_compare_p:nNn {\l_@@_x_initial_dim} = \c_max_dim
        \@@_qpoint:n{col - \l_@@_first_j_tl}
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\pgf@x}
    }
    \bool_if:NT {\l_@@_brace_left_shorten_bool}{
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\c_max_dim}
        \int_step_inline:nnn {\l_@@_first_i_tl}{\l_@@_last_i_tl}{
            \cs_if_exist:cT{pgf @ sh @ ns @\@@_env: - ##1 - \l_@@_first_j_tl}{
                \pgfpointanchor{\@@_env: - ##1 - \l_@@_first_j_tl}{west}
                \dim_set:NN {\l_@@_x_initial_dim}{\pgf@x}
            }
        }
    }
    \bool_lazy_or:nnT{\bool_not_p:n {\l_@@_brace_left_shorten_bool}}{
        \dim_compare_p:nNn {\l_@@_x_initial_dim} = \c_max_dim
        \@@_qpoint:n{col - \l_@@_first_j_tl}
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\pgf@x}
    }
    \bool_if:NT {\l_@@_brace_right_shorten_bool}{
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\c_max_dim}
        \int_step_inline:nnn {\l_@@_first_i_tl}{\l_@@_last_i_tl}{
            \cs_if_exist:cT{pgf @ sh @ ns @\@@_env: - ##1 - \l_@@_last_j_tl}{
                \pgfpointanchor{\@@_env: - ##1 - \l_@@_last_j_tl}{east}
                \dim_set:NN {\l_@@_x_final_dim}{\pgf@x}
            }
        }
    }
    \dim_set_eq:NN {\l_@@_x_final_dim}{\c_max_dim}
    \int_step_inline:nnn {\l_@@_first_i_tl}{\l_@@_last_i_tl}{
        \cs_if_exist:cT{pgf @ sh @ ns @\@@_env: - ##1 - \l_@@_last_j_tl}{
            \pgfpointanchor{\@@_env: - ##1 - \l_@@_last_j_tl}{east}
            \dim_set:NN {\l_@@_x_final_dim}{\pgf@x}
        }
    }
    \pgfsetcorners{\l_@@_x_initial_dim}{\l_@@_x_final_dim}
    \pgfusepath{stroke}
    \pgfresetcornersfalse
    \bool_lazy_or:nnTF{\bool_not_p:n \l_@@_brace_left_shorten_bool}{\dim_compare_p:nNn {\l_@@_x_initial_dim} = \c_max_dim}{
        \@@_qpoint:n{col - \l_@@_first_j_tl}
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\pgf@x}
    }
    \bool_lazy_or:nnT{\bool_not_p:n \l_@@_brace_right_shorten_bool}{
        \dim_compare_p:nNn {\l_@@_x_initial_dim} = \c_max_dim
        \@@_qpoint:n{col - \l_@@_first_j_tl}
        \dim_set_eq:NN {\l_@@_x_initial_dim}{\pgf@x}
    }
}
\end{code}
\bool_lazy_or:nnT \bool_not_p:n { \l_@@_brace_right_shorten_bool } \dim_compare_p:nNNn { \l_@@_x_final_dim } = { - \c_max_dim } \bool_not_p:n \l_@@_brace_right_shorten_bool \dim_set_eq:NN \l_@@_x_final_dim \pgf@x \str_if_eq:nnTF { #5 } { under } { \@@_underbrace_i:n { #3 } } { \@@_overbrace_i:n { #3 } } \endpgfpicture \group_end: \group_end:}

The argument is the text to put above the brace.

\cs_new_protected:Npn \@@_overbrace_i:n #1 { \@@_qpoint:n { row - \l_@@_first_i_tl } \pgftransformshift { \pgfpoint { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 } \pgf@y + \l_@@_brace_yshift_dim } \pgfnode { rectangle } { south } { } { } \vbox_top:n { \group_begin: \everycr { } \halign { \hfil #1 \hfil \crcr \@@_math_toggle_token: #1 \@@_math_toggle_token: \crcr \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip } \hbox_to_wd:nn { \l_@@_x_final_dim - \l_@@_x_initial_dim } { \downbracefill } \cr } \group_end: } { } { } }

The argument is the text to put under the brace.

\cs_new_protected:Npn \@@_underbrace_i:n #1 { \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } } \pgftransformshift { \pgfpoint { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 } \pgf@y - \l_@@_brace_yshift_dim } \pgfnode { rectangle } { } { } { } }
The command \ShowCellNames

\NewDocumentCommand \@@_ShowCellNames { } {
  \dim_zero_new:N \g_@@_tmpc_dim
  \dim_zero_new:N \g_@@_tmpd_dim
  \dim_zero_new:N \g_@@_tmpe_dim
  \int_step_inline:nn \c@iRow
    \begin { pgfpicture }
      \@@_qpoint:n { row - ##1 }
      \dim_set_eq:NN \l_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
      \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
      \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
    \end { pgfpicture }
    \int_step_inline:nn \c@jCol
      \hbox_set:Nn \l_tmpa_box { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
      \begin { pgfpicture }
        \@@_qpoint:n { col - ####1 }
        \dim_gset_eq:NN \l_tempa_dim \pgf@x
        \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
        \dim_gset:Nn \l_tempb_dim { \pgf@x - \g_@@_tmpe_dim }
        \dim_gset_eq:NN \l_tempe_dim \pgf@x
        \end { pgfpicture }
        \fp_set:Nn \l_tmpa_fp { \fp_min:nn } \fp_min:nn
        \fp_min:nn
          \dim_ratio:nn { \g_@@_tmpe_dim } \dim_ratio:nn { \g_@@_tmpe_dim }
        \box_scale:Nn \l_tmpa_fp { \fp_use:N \l_tmpa_fp } \fp_use:N \l_tmpa_fp
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
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 `\c_@@_messages_for_Overleaf_bool` corresponds to the key `messages-for-Overleaf`.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

The boolean `\c_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

The available keys are (in alphabetic order):
- `footnote`
- `footnotehyper`
- `messages-for-Overleaf`
- `renew-dots`
- `renew-matrix`

The boolean `\c_@@_footnotebool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

The available keys are (in alphabetic order):
- `footnote`
- `footnotehyper`
- `messages-for-Overleaf`
- `renew-dots`
- `renew-matrix`

\keys_define:nn { NiceMatrix / Package }
{ `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 ,
`messages-for-Overleaf` .bool_set:N = \c_@@_messages_for_Overleaf_bool ,
`footnote` .bool_set:N = \c_@@_footnote_bool ,
`footnotehyper` .bool_set:N = \c_@@_footnotehyper_bool ,
`unknown` .code:n = \@@_error:n { Unknown-key-for-package } }
\ProcessKeysOptions { NiceMatrix / 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.

The package footnote won't be loaded.

You can't use the option 'footnotehyper' because the package footnote 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 footnotehyper. The package footnotehyper won't be loaded.

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.

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

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

For a list of the available keys, type H.<return>.
If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that’s why we can’t put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

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

The error messages are as follows:

- **Negative weight**
  - Weight of the ‘X’-columns must be positive and you have used the value ‘#1’.
  - The absolute value will be used.

- **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.

- **Too much cols for matrix with last col**
  - The maximal number of columns for a matrix is fixed by the LaTeX counter ‘MaxMatrixCols’. Its actual value is `\int_use:N \c@MaxMatrixCols`. This error is fatal.

- **Too much cols for matrix**
  - In the row `\int_eval:n { \c@jCol - 1 }`, you try to use more columns than allowed by your `\@@_full_name_env:` `\@@_message_hdotsfor:`. Recall that the maximal number of columns for a matrix is fixed by the LaTeX counter ‘MaxMatrixCols’. Its actual value is `\int_use:N \c@MaxMatrixCols`. This error is fatal.
For the following message, remind that the test is not done after the construction of the array but in each row. That’s why we have to put \c@jCol-1 and not \c@jCol.

\@@_msg_new:nn { too-much-cols-for-array }
{
  Too-much-columns.\"
  In-the-row-\int_eval:n { \c@jCol - 1 },-
  -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).-\nThis-error-is-fatal.
}
\@@_msg_new:nn { hvlines-except-corners }
{
  Obsolete-key.\"
  The-key-'hvlines-except-corners'-is-now-obsolete.-You-should-instead-use-the-
  keys-'hvlines'-'and-'corners'.\"
This-error-is-fatal.
}
\@@_msg_new:nn { columns-not-used }
{
  Columns-not-used.\"
  The-preamble-of-your-\@_full_name_env:\ announces-\int_use:N \g_@@_static_num_of_col_int
  columns-but-you-use-only-\int_use:N \c@jCol.\"
  The-columns-you-did-not-used-won't-be-created.
}
\@@_msg_new:nn { in-first-col }
{
  Erroneous-use.\"
  You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\"
  That-command-will-be-ignored.
}
\@@_msg_new:nn { in-last-col }
{
  Erroneous-use.\"
  You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\"
  That-command-will-be-ignored.
}
\@@_msg_new:nn { in-first-row }
{
  Erroneous-use.\"
  You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\"
  That-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.\"
  That-command-will-be-ignored.
}
\@@_msg_new:nn { double-closing-delimiter }
{
  Double-delimiter.\"
  You-can't-put-a-second-closing-delimiter-"#1"-just-after-a-first-closing-
  delimiter.-This-delimiter-will-be-ignored.
}
\@@_msg_new:nn { delimiter-after-opening }
{
  Double-delimiter.\"
  You-can't-put-a-second-delimiter-"#1"-just-after-a-first-opening-
  delimiter.-That-delimiter-will-be-ignored.
}
Bad line style. Since you haven't loaded Tikz, the only value you can give to 'line-style' is 'standard'. That key will be ignored.

Unknown key - 'l_keys_key_str' is unknown in a 'custom-line'. It will be ignored.

The available keys are (in alphabetic order): color, command, dotted, letter, multiplicity, sep-color, tikz, and tota-width.

Unknown key.
As for now, there is only five keys available here: 'color', 'inter', 'line-style', 'radius', and 'shorten' (and you try to use '\l_keys_key_str'). That key will be ignored.

Unknown key.
As for now, there is only two keys available here: 'cols' and 'respect-blocks' (and you try to use '\l_keys_key_str'). That key will be ignored.

Construct too large. Your command \token_to_str:N #1 can't be drawn because your matrix is too small. That command will be ignored.

Ampersand forbidden.
You can't use an ampersand (\token_to_str:N &) to separate columns because the key 'light-syntax' is in force. This error is fatal.

Double-backslash forbidden.
You can't use a double-backslash to separate rows because the key 'light-syntax' is in force. You must use the character '\l_@@_end_of_row_tl' (set by the key 'end-of-row'). This error is fatal.
\@@_msg_new:nn { bad-value-for-baseline }
\{
\Bad-value-for-baseline.\\%
\The-value-given-to-‘baseline’-(\int_use:N \l_tmpa_int)-is-not-
\valid.-The-value-must-be-between-\int_use:N \l_@@_first_row_int\ and-
\int_use:N \g_@@_row_total_int\ or-equal-to-‘t’,-‘c’-or-‘b’-or-of-
the-form-‘line-i’.-\%
\A-value-of-1-will-be-used.
\}
\@@_msg_new:nn { Invalid-name }
\{
\Invalid-name.\%
\You-can’t-give-the-name-‘\l_keys_value_tl’-to-a-\token_to_str:N
\SubMatrix\ of-your-\@@_full_name_env:.\%
\A-name-must-be-accepted-by-the-regular-expression-[A-Za-z][A-Za-z0-9]*.\%
\This-key-will-be-ignored.
\}
\@@_msg_new:nn { Wrong-line-in-SubMatrix }
\{
\Wrong-line.\%
\You-try-to-draw-a-‘#1-line-of-number-‘#2’-in-a-
\token_to_str:N \SubMatrix\ of-your-\@@_full_name_env: but-that-
number-is-not-valid.-It-will-be-ignored.
\}
\@@_msg_new:nn { Impossible-delimiter }
\{
\Impossible-delimiter.\%
\It’s-impossible-to-draw-the-‘#1-delimiter-of-your-
\token_to_str:N \SubMatrix\ because-all-the-cells-are-empty-
in-that-column.\%
\\bool_if:NT \l_@@_submatrix_slim_bool
\{ ~Maybe-you-should-try-without-the-key-‘slim’. \%
\This-\token_to_str:N \SubMatrix\ will-be-ignored.
\}
\@@_msg_new:nn { width-without-X-columns }
\{
\No-X-column.\%
\You-have-used-the-key-‘width’-but-you-have-put-no-‘X’-column. \%
That-key-will-be-ignored.
\}
\@@_msg_new:nn { key-multiplicity-with-dotted }
\{
\Incompatible-keys. \%
\You-have-used-the-key-‘multiplicity’-with-the-key-‘dotted’-
in-a-‘custom-line’.-They-are-incompatible. \%
The-key-‘multiplicity’-will-be-discarded.
\}
\@@_msg_new:nn { empty-environment }
\{
\Empty-environment.\%
\Your-\@@_full_name_env: is-empty.-This-error-is-fatal.
\}
\@@_msg_new:nn { Wrong-use-of-v-center }
\{
\Wrong-use-of-v-center \%
\You-should-not-use-the-key-‘v-center’-here-because-your-block-is-not-
mono-row.-However,-you-can-go-on.
\}
\@@_msg_new:nn { No-letter-and-no-command }
\{
Erroneous use.\]

Your use of 'custom-line' is no-op since you don't have used the key 'letter' (for a letter for vertical rules) nor the key 'command' (to draw horizontal rules).\]

However, you can go on.\]

}\]
\@@_msg_new:nn { Forbidden-letter } \]
{ \underline { Forbidden-letter } \]
You can't use the letter '\l@@_letter_str' for a customized line.\]
It will be ignored.\]

}\]
\@@_msg_new:nn { Several-letters } \]
{ \underline { Wrong-name } \]
You must use only one letter as value for the key 'letter' (and you have used '\l@@_letter_str').\]
It will be ignored.\]

}\]
\@@_msg_new:nn { Delimiter-with-small } \]
{ \underline { Delimiter-forbidden } \]
You can't put a delimiter in the preamble of your \@@_full_name_env: because the key 'small' is in force.\]
This error is fatal.\]

}\]
\@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter } \]
{ \underline { Unknown-cell } \]
Your command \token_to_str:N \line\{#1\}\{#2\} in the \token_to_str:N \CodeAfter\ of your \@@_full_name_env: can't be executed because a cell doesn't exist.\]
This command \token_to_str:N \line will be ignored.\]

}\]
\@@_msg_new:nnn { Duplicate-name-for-SubMatrix } \]
{ \underline { Duplicate-name } \]
The name '#1' is already used for a \token_to_str:N \SubMatrix in this \@@_full_name_env: This key will be ignored.\]
\bool_if:NF \c_@@_messages_for_Overleaf_bool { For a list of the names already used, type H <return>. } \]

}\]
\@@_msg_new:nn { r-or-l-with-preamble } \]
{ \underline { Errorneous-use } \]
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: This key will be ignored.\]

}\]
\@@_msg_new:nn { Hdotsfor-in-col-0 } \]
{ \underline { Errorneous-use } \]
You can't use \token_to_str:N \Hdotsfor in an exterior column of the array. This error is fatal.\]

}
7826 \@@_msg_new:nn { bad-corner } 
7827 \{ \n7828 Bad-corner.\ 
7829 \ld_\textit{#1-is-an-incorrect-specification-for-a-corner-(in-the-key-} \n7830 \'corners').-The-available-values-are:-NW,-SW,-NE-and-SE.\ 
7831 This-specification-of-corner-will-be-ignored. \n7832 \} 
7833 \@@_msg_new:nn { bad-border } 
7834 \{ \n7835 Bad-border.\ 
7836 \ld_\textit{\backslash_key\_key\textit{\_str}\_space-is-an-incorrect-specification-for-a-border-} \n7837 \(\text{in-the-key}'borders'-of-the-command'\texttt{\textit{\_token\_to\_str:N}} \texttt{\textit{\_Block})}.- \n7838 The-available-values-are:-left,-right,-top-and-bottom-(and-you-can-} \n7839 \ld_\textit{also-use-the-key}'tikz' \n7840 \ld_\textit{\texttt{\textit{\_bool\_if:N}} \texttt{\_c_@@_tikz\_loaded\_bool} \n7841 \{if-you-load-the-LaTeX-package-'tikz'}\}).\ 
7842 This-specification-of-border-will-be-ignored. \n7843 \} 
7844 \@@_msg_new:nn { tikz-key-without-tikz } 
7845 \{ \n7846 Tikz-not-loaded.\ 
7847 You-can't-use-the-key-'tikz'-for-the-command-'\texttt{\textit{\_token\_to\_str:N}} \texttt{\textit{\_Block})-because-you-have-not-loaded-Tikz.- \n7848 This-key-will-be-ignored. \n7849 \} 
7850 \@@_msg_new:nn { last-col-non-empty-for-NiceArray } 
7851 \{ \n7852 Erroneous-use.\ 
7853 In-the-'\@@_full_name_env:'-you-must-use-the-key- \n7854 'last-col'-without-value.\ 
7855 However,-you-can-go-on-for-this-time- \n7856 (the-value-'\ld_\textit{\backslash_key\_value\_tl}'-will-be-ignored). \n7857 \} 
7858 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions } 
7859 \{ \n7860 Erroneous-use.\ 
7861 In-\ld_\textit{NiceMatrixOptions},-you-must-use-the-key- \n7862 'last-col'-without-value.\ 
7863 However,-you-can-go-on-for-this-time- \n7864 (the-value-'\ld_\textit{\backslash_key\_value\_tl}'-will-be-ignored). \n7865 \} 
7866 \@@_msg_new:nn { Block-too-large-1 } 
7867 \{ \n7868 Block-too-large.\ 
7869 You-try-to-draw-a-block-in-the-cell-\ld_\textit{\_1}\-\ld_\textit{\_2}-of-your-matrix-but-the-matrix-is- \n7870 too-small-for-that-block. \ 
7871 \} 
7872 \@@_msg_new:nn { Block-too-large-2 } 
7873 \{ \n7874 Block-too-large.\ 
7875 The-preamble-of-your-'\@@_full_name_env:'-announces-'\texttt{\textit{\_int\_use:N}} \n7876 \ld_\texttt{\textit{g_@@_static\_num\_of\_col\_int}} \n7877 columns-but-you-use-only-'\texttt{\textit{\_int\_use:N}} \texttt{\textit{\_c@jCol}} \texttt{\textit{\_and-that's-why-a-block-} \n7878 specified-in-the-cell-\ld_\texttt{\textit{\_1}}-\ld_\texttt{\textit{\_2}}-can't-be-drawn.-You-should-add-some-ampersands-} \n7879 (&)-at-the-end-of-the-first-row-of-your- \n7880 \ld_\texttt{\textit{\_@@_full_name_env:}}\ .\ 
7881 This-block-and-maybe-others-will-be-ignored. \n7882 \} 
7883 \@@_msg_new:nn { unknown-column-type } 
7884 \{ \n7885 Bad-column-type. \ 
7886 \}
The column type '#1'in your \@_full_name_env:
is unknown. \\This error is fatal.
}
\@_msg_new:nn { tabularnote-forbidden }
{ Forbidden-command.\}
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\}. \\This command will be ignored.
}
\@_msg_new:nn { borders-forbidden }
{ Forbidden-key.\}
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. \\This key will be ignored.
}
\@_msg_new:nn { bottomrule-without-booktabs }
{ booktabs-not-loaded.\}
You can't use the key 'tabular/bottomrule' because you haven't loaded 'booktabs'. \\This key will be ignored.
}
\@_msg_new:nn { enumitem-not-loaded }
{ enumitem-not-loaded.\}
You can't use the command \token_to_str:N \tabularnote
-because you haven't loaded 'enumitem'. \\This command will be ignored.
}
\@_msg_new:nn { tikz-in-custom-line-without-tikz }
{ \Tikz\not-loaded.\}
You have used the key 'tikz'-in-the-definition-of-a-
customized-line (with 'custom-line') but Tikz is not loaded. You can-continue but you will have another error if you actually-
use that custom line.
}
\@_msg_new:nn { tikz-in-borders-without-tikz }
{ \Tikz\not-loaded.\}
You have used the key 'tikz'-in-a-key 'borders'-(of-a-
command \token_to_str:N \Block) but Tikz is not loaded. \That key will be ignored.
}
\@_msg_new:nn { color-in-custom-line-with-tikz }
{ \Erroneous-use.\}
In a 'custom-line', you have used both 'tikz' and 'color',-
which is forbidden (you should use 'color'-inside-the-key 'tikz').
The key 'color' will be discarded.
}
\@_msg_new:nn { Wrong-last-row }
{ \Wrong-number.\}
You have used 'last-row=\int_use:N \l_@@_last_row_int'-but-your-
7947 \@@_full_name_env:\ seems-to-have\int_use:N \c@iRow \ rows.~
7948 If-you-go-on,-the-value-of\int_use:N \c@iRow \ will-be-used-for-
7949 last-row.~You-can-avoid-this-problem-by-using-'last-row'~
7950 without-value-(more-compilations-might-be-necessary).
7951 }
7952 \@@_msg_new:nn { Yet-in-env }
7953 {
7954 Nested-environments.\
7955 Environments-of-nicematrix-can't-be-nested.\
7956 This-error-is-fatal.
7957 }
7958 \@@_msg_new:nn { Outside(math-mode) }
7959 {
7960 Outside(math-mode.\
7961 The-\@@_full_name_env:\ can-be-used-only-in-math-mode-
7962 (and-not-in-\token_to_str:N \vcenter).\
7963 This-error-is-fatal.
7964 }
7965 \@@_msg_new:nn { One-letter-allowed }
7966 {
7967 Bad-name.\
7968 The-value-of-key-‘l_keys_key_str’-must-be-of-length-1.\
7969 It-will-be-ignored.
7970 }
7971 \@@_msg_new:nn { varwidth-not-loaded }
7972 {
7973 varwidth-not-loaded.\
7974 You-can’t-use-the-column-type-'V'-because-'varwidth'-is-not-
7975 loaded.\
7976 Your-column-will-behave-like-'p'.
7977 }
7978 \@@_msg_new:nnn { Unknown-key-for-Block }
7979 {
7980 Unknown-key.\
7981 The-key-‘l_keys_key_str’-is-unknown-for-the-command-\token_to_str:N \Block.\
7982 It-will-be-ignored. \c_@@_available_keys_str
7983 }
7984 {
7985 The-available-keys-are-(in-alphabetic-order):-b,-borders,-c,-draw,-fill,-
7986 hlines,-hlines,-l,-line-width,-name,-rounded-corners,-r,-respect arraystretch,
7987 -t,-tikz-and-vlines.
7988 }
7989 \@@_msg_new:nn { Version-of-siunitx-too-old }
7990 {
7991 siunitx-too-old.\
7992 You-can’t-use-'S'-columns-because-your-version-of-‘siunitx’-
7993 is-too-old.-You-need-at-least-v-3.0.\
7994 This-error-is-fatal.
7995 }
7996 \@@_msg_new:nnn { Unknown-key-for-Brace }
7997 {
7998 Unknown-key.\
7999 The-key-‘l_keys_key_str’-is-unknown-for-the-commands-\token_to_str:N \UnderBrace\ and-\token_to_str:N \OverBrace.\
8000 It-will-be-ignored. \c_@@_available_keys_str
8001 }
8002 {
8003 The-available-keys-are-(in-alphabetic-order):-color,-left-shorten,-
8004 right-shorten,-shorten-(which-fixes-both-left-shorten-and-
right-shorten)-and-yshift.

\@@_msg_new:nnn { Unknown-key-for-CodeAfter }
{
  Unknown-key.\"
  The-key-\"l_keys_key_str\"-is-unknown.\"
  It-will-be-ignored. \"
  \c_@@_available_keys_str
}
{
  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 }
{
  Unknown-key.\"
  The-key-\"l_keys_key_str\"-is-unknown.\"
  That-key-will-be-ignored. \"
  \c_@@_available_keys_str
}
{
  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 }
{
  Unknown-key.\"
  The-key-\"l_keys_key_str\"-is-unknown.\"
  That-key-will-be-ignored. \"
  \c_@@_available_keys_str
}
{
  The-available-keys-are-(in-alphabetic-order):
  bottomrule,-
  code-after,-
  code-before,-
  detect-duplicates,-
  enumitem-keys,-
  enumitem-keys-para,-
  para,-
  label-in-list,-
  label-in-tabular-and-
  style.
}

\@@_msg_new:nnn { Unknown-key-for-RowStyle }
{
  Unknown-key.\"
  The-key-\"l_keys_key_str\"-is-unknown-for-the-command-
That key will be ignored.

The available keys are (in alphabetic order):
'bold',
'cell-space-top-limit',
'cell-space-bottom-limit',
'cell-space-limits',
'color',
'nb-rows'-and-
'rowcolor'.

Unknown key for \NiceMatrixOptions

The key '\l_keys_key_str' is unknown for the command \NiceMatrixOptions.
That key will be ignored.

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,
corners,
custom-key,
create-extra-nodes,
create-medium-nodes,
create-large-nodes,
delimiters-(several-subkeys),
end-of-row,
first-col,
first-row,
\hlines,
\hlines,
last-col,
last-row,
left-margin,
light-syntax,
notes-(several-subkeys),
nullify-dots,
renew-dots,
renew-matrix,
respect-arraystretch,
right-margin,
rules-(with-the-subkeys-'color'-and-'width'),
small,
sub-matrix-(several-subkeys),
\vlines,
xdots-(several-subkeys).

Unknown key for \NiceArray

The key '\l_keys_key_str' is unknown for the environment-
That key will be ignored.

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,
- colorbl-like,
- columns-width,
- corners,
- create-extra-nodes,
- create-medium-nodes,
- create-large-nodes,
- delimiters/color,
- extra-left-margin,
- extra-right-margin,
- first-col,
- first-row,
- hlines,
- hvlines,
- last-col,
- last-row,
- left-margin,
- light-syntax,
- name,
- notes/bottomrule,
- notes/para,
- nullify-dots,
- renew-dots,
- respect-arraystretch,
- right-margin,
- rules-(with-the-subkeys-'color'-and-'width')
- small,
- t,
- tabularnote,
- vlines,
- xdots/color,
- xdots/shorten-start,
- xdots/shorten-end,
- xdots/shorten-and-
- xdots/line-style.

This error message is used for the set of keys \texttt{NiceMatrix/NiceMatrix} and \texttt{NiceMatrix/pNiceArray} (but not by \texttt{NiceMatrix/NiceArray} because, for this set of keys, there is also the keys t, c and b).
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,
colorbl-like,
columns-width,
corners,
create-extra-nodes,
create-medium-nodes,
create-large-nodes,
delimiters-(several-subkeys),
extra-left-margin,
extra-right-margin,
first-col,
first-row,
hlines,
hvlines,
l,
last-col,
last-row,
left-margin,
light-syntax,
name,
nullify-dots,
r,
renew-dots,
respect-arraystretch,
right-margin,
rules-(with-the-subkeys-'color'-and-'width'),
small,
t,
vlines,
xdots/color,
xdots/shorten-start,
xdots/shorten-end,
xdots/shorten-and,
xdots/line-style.

\@@_msg_new:nnn { Unknown-key-for-NiceTabular }

{ Unknown-key.\
The-key-"\$l_keys_key_str"-is-unknown-for-the-environment-
\{NiceTabular\}. \|
That-key-will-be-ignored. \|
\c_@@_available_keys_str
}

{ The-available-keys-are-(in-alphabetic-order):

b, baseline,
c, cell-space-bottom-limit,
cell-space-limits,
cell-space-top-limit,
code-after,
\@@_msg_new:nnn { Duplicate-name }\}
{ Duplicate-name.\}
The-name-'\l_keys_value_tl'-is-already-used-and-you-shouldn't-use-
the-same-environment-name-twice.-You-can-go-on,-but,-
maybe,-you-will-have-incorrect-results-especially-
if-you-use-'columns-width=auto'.-If-you-don't-want-to-see-this-
message-again,-use-the-key-'allow-duplicate-names'-in-
'\token_to_str:N \NiceMatrixOptions'.\}
\c_@@_available_keys_str\}
\@@_msg_new:nn { Option-auto-for-columns-width }\}
{ Erroneous-use.\}
You-can't-give-the-value-'auto'-to-the-key-'columns-width'-here.-
That-key-will-be-ignored.
20 History

The successive versions of the file nicematrix.sty provided by TeXLive are available on the svn server of TeXLive:
https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment \texttt{\textbackslash NiceArray} with column types L, C and R.

Changes between version 1.2 and 1.3

New environment \texttt{\textbackslash pNiceArrayC} and its variants.
Correction of a bug in the definition of \texttt{\textbackslash BNiceMatrix}, \texttt{\textbackslash vNiceMatrix} and \texttt{\textbackslash VNiceMatrix} (in fact, it was a typo).
Options are now available locally in \texttt{\textbackslash pNiceMatrix} and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types \texttt{w} and \texttt{W} can now be used in the environments \texttt{\textbackslash NiceArray}, \texttt{\textbackslash pNiceArrayC} and its variants with the same meaning as in the package \texttt{array}.
New option \texttt{columns-width} to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of nicematrix were focused on the continuous dotted lines whereas the version 2.0 of nicematrix provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment \texttt{\textbackslash pNiceArrayRC}. With this new implementation, there is no restriction on the width of the columns.
The package nicematrix no longer loads mathtools but only amsmath.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option \texttt{code-for-first-row} is now available in the command \texttt{\textbackslash NiceMatrixOptions}.
Following a discussion on TeX StackExchange\footnote{\texttt{\textbackslash tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}}, Tikz externalization is now deactivated in the environments of the package nicematrix.\footnote{Before this version, there was an error when using nicematrix with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by nicematrix because they use the options \texttt{overlay} and \texttt{remember picture}.}
Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like \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 \Cdots command (and a single \Vdots).

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

Changes between version 2.1.3 and 2.1.4

Replacement of some options 0 \{ \} in commands and environments defined with xparse by ! 0 \{ \} (because a recent version of xparse introduced the specifier ! and modified the default behaviour of the last optional arguments).


Changes between version 2.1.4 and 2.1.5

Compatibility with the classes revtex4-1 and revtex4-2.
Option allow-duplicate-names.

Changes between version 2.1.5 and 2.2

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

Changes between version 2.2 and 2.2.1

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

Changes between version 2.2.1 and 2.3

Compatibility with the column type S of siunitx.
Option hlines.

Changes between version 2.3 and 3.0

Modification of \Hdots. Now \Hdots erases the \vlines (of “|”) as \hdots 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 \cddottedline, the symbol “:” (in the preamble of the array) and \line in code-after).
The starred versions of \Dots, \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 colorbl.
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 preamble of an array when 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 previous version of nicematrix was incompatible with a recent version of expl3 (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange\footnote{cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize}, optimization of Tikz externalization is disabled in the environments of nicematrix when the class standalone or the package standalone is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the code-after was not executed.

Changes between version 3.5 and 3.6

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

Changes between version 3.6 and 3.7

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

\footnote{cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize}
Changes between version 3.7 and 3.8

New programmation for the command $\texttt{\Block}$ when the block has only one row. With this programmation, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with $\texttt{\multicolumn}$. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands $\texttt{\NiceMatrixLastEnv}$ and $\texttt{\OnlyMainNiceMatrix}$. New options create-medium-nodes and 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 $\texttt{baseline}$ for $\texttt{\NiceArray}$ (not for the other environments).

The name of the Tikz nodes created by the command $\texttt{\Block}$ has changed: when the command has been issued in the cell $i$-$j$, the name is $i$-$j$-block and, if the creation of the “medium nodes” is required, a node $i$-$j$-block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by 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 $1$ ($=\texttt{L}$) or $r$ ($=\texttt{R}$) 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 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 $\texttt{pgf}$. By default, Tikz is not loaded by $\texttt{nicematrix}$.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on stackoverflow).

Better error messages when the user uses $\&$ or $\texttt{\textbackslash\textbackslash}$ 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 \Ldots, \Cdots, \Vdots, \Ddots, \Iddots, \Hdotsfor and the command \line in the code-after with the tokens _ and ^.
The option baseline is now available in all the environments of nicematrix. Before, it was available only in \texttt{NiceArray}.
New keyword \CodeAfter (in the environments of 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 cell-space-top-limit and cell-space-bottom-limit
New command \texttt{diagbox}
The key 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\\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{!{\qquad}} 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 hvlines-except-corners.

Changes between versions 4.4 and 5.0

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

Changes between versions 5.0 and 5.1

The vertical rules specified by | in the preamble are not broken by \texttt{\hline\hline} (and other).
Environment \texttt{NiceTabular*}
Command \texttt{\Ddotsfor} similar to \texttt{Hdotsfor}
The variable \texttt{\g_nicematrix_code_after_tl} is now public.
Changes between versions 5.1 and 5.2

The vertical rules specified by | or || in the preamble respect the blocks. Key \texttt{respect-blocks} for \texttt{rowcolors} (with a s) in the \texttt{code-before}. The variable \texttt{g_nicematrix_code_before_tl} is now public. The key \texttt{baseline} may take in as value an expression of the form \texttt{line-i} to align the \texttt{\hline} in the row \texttt{i}. The key \texttt{hvlines-except-corners} may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

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

Changes between versions 5.3 and 5.4

Key \texttt{tabularnote}. Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put \texttt{\omit} before \texttt{\CodeAfter}. Correction of a bug: the tabular notes \texttt{\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 \texttt{\NotEmpty}.

Changes between versions 5.6 and 5.7

New key \texttt{delimiters-color} Keys \texttt{fill}, \texttt{draw} and \texttt{line-width} for the command \texttt{\Block}.

Changes between versions 5.7 and 5.8

Keys \texttt{cols} and \texttt{restart} of the command \texttt{\rowcolors} in the \texttt{code-before}. Modification of the behaviour of \texttt{\ } in the columns of type \texttt{p}, \texttt{m} or \texttt{b} (for a behaviour similar to the environments of \texttt{array}). Better error messages for the command \texttt{\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 \texttt{line-style} for the continuous dotted lines when the Tikz library \texttt{babel} was loaded. New key \texttt{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}.
Changes between versions 5.10 and 5.11

It’s now possible, in the `code-before` and in the `\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 `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).
New key `delimiters/max-width`.
New keys `hlines`, `vlines` and `hlines` for the command `\SubMatrix` in the `\CodeAfter`.
New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).
New key `borders` for the command `\Block`.
New command `\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).

Nodes of the form `(1.5), (2.5), (3.5), etc.
Keys `t` and `b` for the command `\Block`.
Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.
The commands provided by `nicematrix` to color cells, rows and columns don’t color the cells which are in the “corners” (when the key `corner` is used).
It’s now possible to specify delimiters for submatrices in the preamble of an environment.
The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It’s now possible to use the cells corresponding to the contents of the nodes (of the form i–j) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.
Keys `L`, `C` and `R` for the command `\Block`.
Key `hvlines-except-borders`.
It’s now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`. 247
Changes between versions 5.19 and 6.0

Columns \texttt{X} and environment \texttt{\{NiceTabularX\}}.
Command \texttt{\rowlistcolors} available in the \texttt{\CodeBefore}.
In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).
The key \texttt{define-L-C-R} has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the \texttt{X} columns.
Key \texttt{color} for the command \texttt{\RowStyle}.

Changes between versions 6.1 and 6.2

Better compatibility with the classes \texttt{revtex4-1} and \texttt{revtex4-2}.
Key \texttt{vlines-in-sub-matrix}.

Changes between versions 6.2 and 6.3

Keys \texttt{nb-rows}, \texttt{rowcolor} and \texttt{bold} for the command \texttt{\RowStyle}
Key \texttt{name} for the command \texttt{\Block}.
Support for the columns \texttt{V} of \texttt{varwidth}.

Changes between versions 6.3 and 6.4

New commands \texttt{\UnderBrace} and \texttt{\OverBrace} in the \texttt{\CodeAfter}.
Correction of a bug of the key \texttt{baseline} (cf. question 623258 on TeX StackExchange).
Correction of a bug with the columns \texttt{V} of \texttt{varwidth}.
Correction of a bug: the use of \texttt{\hdottedline} and \texttt{\:} in the preamble of the array (of another letter specified by \texttt{letter-for-dotted-lines}) was incompatible with the key \texttt{xdots/line-style}.

Changes between versions 6.4 and 6.5

Key \texttt{custom-line} in \texttt{\NiceMatrixOptions}.
Key \texttt{respect-arraystretch}.

Changes between version 6.5 and 6.6

Keys \texttt{tikz} and \texttt{width} in \texttt{custom-line}.

Changes between version 6.6 and 6.7

Key \texttt{color} for \texttt{\OverBrace} and \texttt{\UnderBrace} in the \texttt{\CodeAfter}
Key \texttt{tikz} in the key \texttt{borders} of a command \texttt{\Block}

Changes between version 6.7 and 6.8

In the notes of a tabular (with the command \texttt{\tabularnote}), the duplicates are now detected: when several commands \texttt{\tabularnote} are used with the same argument, only one note is created at the end of the tabular (but all the labels are present, of course).
Changes between version 6.8 and 6.9

New keys \dots/shorten-start and \dots/shorten-end for customisation of the continuous dotted lines.

New command \ShowCellNames available in the \CodeBefore and in the \CodeAfter.

Changes between version 6.9 and 6.10

New keys \dots/shorten-start and \dots/shorten-end.

It’s possible to use \line in the \CodeAfter between two blocks (and not only two cells).

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.

Symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_l_@NiceTabular_bool</td>
<td>186, 248, 913, 1101, 1408, 1411, 1539, 1677, 1681, 1775, 1787, 2788, 3006, 3026, 3035, 5893, 5905, 6697</td>
</tr>
<tr>
<td>_l_@NotEmpty</td>
<td>1268, 2998</td>
</tr>
<tr>
<td>_l_@OnlyMainNiceMatrix_i:n</td>
<td>1264, 4784</td>
</tr>
<tr>
<td>_l_@OnlyMainNiceMatrix_i:n</td>
<td>4787, 4794, 4797</td>
</tr>
<tr>
<td>_l_@OverBrace</td>
<td>3152, 7248</td>
</tr>
<tr>
<td>_l_@RowStyle:n</td>
<td>1269, 4380</td>
</tr>
<tr>
<td>_l_@S</td>
<td>217, 1817</td>
</tr>
<tr>
<td>_l_@ShowCellNames</td>
<td>1401, 3153, 7400</td>
</tr>
<tr>
<td>_l_@SubMatrix</td>
<td>3150, 6938</td>
</tr>
<tr>
<td>_l_@SubMatrix_in_code_before</td>
<td>1400, 6918</td>
</tr>
<tr>
<td>_l_@UnderBrace</td>
<td>3151, 7243</td>
</tr>
<tr>
<td>_l_@V</td>
<td>1732, 1813</td>
</tr>
<tr>
<td>_l_@Vdots</td>
<td>1184, 1255, 4044</td>
</tr>
<tr>
<td>_l_@Vdotsfor_lines_tl</td>
<td>1287, 3215</td>
</tr>
<tr>
<td>_l_@Vdotsfor:nnnn</td>
<td>1261, 4219</td>
</tr>
<tr>
<td>_l_@Vdotsfor:nnnn</td>
<td>4223, 4234</td>
</tr>
<tr>
<td>_l_@Vdotsfor:nnnn</td>
<td>1731, 1816, 2310</td>
</tr>
<tr>
<td>_l_@X</td>
<td>1825, 3021</td>
</tr>
<tr>
<td>_l_@X_columns_dim</td>
<td>253, 2210, 8587</td>
</tr>
<tr>
<td>_l_@X_columns_aux_bool</td>
<td>282, 1597, 2199</td>
</tr>
<tr>
<td>_l_@X_columns_dim</td>
<td>283, 1598, 1607, 1613, 2202</td>
</tr>
<tr>
<td>_l_@actual_color:</td>
<td>1410, 4469</td>
</tr>
<tr>
<td>_l_@actual_diagbox:nnnnnnn</td>
<td>5870, 6213, 6709, 6726</td>
</tr>
<tr>
<td>_l_@actually_draw_Cdots:</td>
<td>3584, 3588</td>
</tr>
<tr>
<td>_l_@actually_draw_Ddots:</td>
<td>3734, 3738</td>
</tr>
<tr>
<td>_l_@actually_draw_Iddots:</td>
<td>3782, 3786</td>
</tr>
<tr>
<td>_l_@actually_draw_Ldots:</td>
<td>3754, 3549, 4210</td>
</tr>
<tr>
<td>_l_@actually_draw_Vdots:</td>
<td>3666, 3670, 4285</td>
</tr>
<tr>
<td>_l_@add_to_colors_seq:nn</td>
<td>4455, 4467, 4468, 4492, 4501, 4510, 4519, 4623</td>
</tr>
<tr>
<td>_l_@adjust_pos_of_blocks_seq:</td>
<td>3134, 3180</td>
</tr>
<tr>
<td>_l_@adjust_pos_of_blocks_seq_i:nnnnn</td>
<td>3183, 3185</td>
</tr>
<tr>
<td>_l_@adjust_size_box:</td>
<td>986, 1013, 2061, 2370, 2885, 2930</td>
</tr>
<tr>
<td>_l_@adjust_to_submatrix:nn</td>
<td>3429, 3532, 3571, 3652, 3727, 3775</td>
</tr>
<tr>
<td>_l_@adjust_to_submatrix:nnnnnn</td>
<td>3436, 3438</td>
</tr>
<tr>
<td>_l_@after_array:</td>
<td>1713, 3039</td>
</tr>
<tr>
<td>_l_@after_col_zero_bool</td>
<td>284, 1150, 2862, 4126</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
4.1 General case 4
4.2 The mono-column blocks 6
4.3 The mono-row blocks 6
4.4 The mono-cell blocks 6
4.5 Horizontal position of the content of the block 7
5 The rules 7
5.1 Some differences with the classical environments 8
5.1.1 The vertical rules 8
5.1.2 The command \vline 8
5.2 The thickness and the color of the rules 9
5.3 The tools of nicematrix for the rules 9
5.3.1 The keys hlines and vlines 9
5.3.2 The keys hvlines and hvlines-except-borders 10
5.3.3 The (empty) corners 10
17 Technical remarks

17.1 Diagonal lines .................................................. 45
17.2 The empty cells .................................................. 46
17.3 The option exterior-arraycolsep .............................. 46
17.4 Incompatibilities .................................................. 47

18 Examples

18.1 Utilisation of the key 'tikz' of the command \Block ........ 47
18.2 Notes in the tabulars ............................................. 48
18.3 Dotted lines ..................................................... 49
18.4 Dotted lines which are no longer dotted .................... 50
18.5 Dashed rules ..................................................... 51
18.6 Stacks of matrices ................................................ 51
18.7 How to highlight cells of a matrix ............................ 55
18.8 Utilisation of \SubMatrix in the \CodeBefore ............... 57

19 Implementation ................................................. 58

20 History ............................................................ 241

Index ............................................................... 249