

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 23, 2021

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.

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

Product	dimensions (cm)			Price
	L	l	h	
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

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, TeXLive 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`.¹

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**.²

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 5.19 of `nicematrix`, at the date of 2021/07/23.

¹The 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>

²If 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.

<code>{NiceTabular}</code>	<code>{NiceArray}</code>	<code>{NiceMatrix}</code>
<code>{NiceTabular*}</code>	<code>{pNiceArray}</code>	<code>{pNiceMatrix}</code>
	<code>{bNiceArray}</code>	<code>{bNiceMatrix}</code>
	<code>{BNiceArray}</code>	<code>{BNiceMatrix}</code>
	<code>{vNiceArray}</code>	<code>{vNiceMatrix}</code>
	<code>{VNiceArray}</code>	<code>{VNiceMatrix}</code>

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.

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.**

Important

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

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

This key will probably be deleted in a future version of `nicematrix`.

2 The vertical space between the rows

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

```
\begin{pmatrix}
\frac{1}{2} & -\frac{1}{2} \\
\frac{1}{3} & \frac{1}{4} \\
\end{pmatrix}
```

$$\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{3} & \frac{1}{4} \end{pmatrix}$$

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`.³

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

³One 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`.

```

 $\begin{pNiceMatrix}$ 
 $\frac{1}{2}$  &  $-\frac{1}{2}$  \\
 $\frac{1}{3}$  &  $\frac{1}{4}$  \\
 $\end{pNiceMatrix}$ 

```

$$\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{3} & \frac{1}{4} \end{pmatrix}$$

3 The vertical position of the arrays

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

```

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

```

$$A = \begin{pmatrix} \frac{1}{\sqrt{1+p^2}} & p & 1-p \\ 1 & 1 & 1 \\ 1 & p & 1+p \end{pmatrix}$$

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

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

```

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

```

1. an item

$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

However, it's also possible to use the tools of `booktabs`⁴: `\toprule`, `\bottomrule`, `\midrule`, etc.

```

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

```

1. an item

$$\begin{array}{cccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ u_n & 1 & 2 & 4 & 8 & 16 & 32 \end{array}$$

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

```

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

```

⁴The extension `booktabs` is *not* loaded by `nicematrix`.

```

$A=\begin{pNiceArray}{cc|cc}[baseline=line-3]
\dfrac{1}{A} & \dfrac{1}{B} & 0 & 0 \\
\dfrac{1}{C} & \dfrac{1}{D} & 0 & 0 \\
\hline
0 & 0 & A & B \\
0 & 0 & D & D \\
\end{pNiceArray}$

```

$$A = \left(\begin{array}{cc|cc} \frac{1}{A} & \frac{1}{B} & 0 & 0 \\ \frac{1}{C} & \frac{1}{D} & 0 & 0 \\ \hline 0 & 0 & A & B \\ 0 & 0 & D & D \end{array} \right)$$

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.⁵

The command `\Block` must be used in the upper leftmost cell of the array with two arguments.

- The first argument is the size of the block with the syntax i - j where i is the number of rows of the block and j its number of columns.

If this argument is empty, its default value is 1-1. If the number of rows is not specified, or equal to *, the block extends until the last row (idem for the columns).

- The second argument is the content of the block. It's possible to use `\\` in that content to have a content on several lines. In `{NiceTabular}` the content of the block is composed in text mode whereas, in the other environments, it is composed in math mode.

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

```

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

```

$$\left[\begin{array}{ccc|c} A & & & 0 \\ & \hspace*{1cm} & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.⁶

```

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

```

$$\left[\begin{array}{ccc|c} A & & & 0 \\ & \hspace*{1cm} & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

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

⁵The spaces after a command `\Block` are deleted.

⁶This argument between angular brackets may also be used to insert a command of font such as `\bfseries` when the command `\\` is used in the content of the block.

```

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

```

$$\left[\begin{array}{ccc|c} & & & 0 \\ & & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

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

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `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 `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 key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁷);
- the key `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 `left`, `right`, `top` and `bottom`;
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hvlines` draws all the vertical and horizontal rules in the block;
- **New 5.19** when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁸ by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 41.

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

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

```

\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose      & tulipe & marguerite & dahlia \\
violette  & & & \\
& \Block[draw=red,fill=[RGB]{204,204,255},rounded-corners]{2-2}
& \LARGE De très jolies fleurs}
& & souci \\
pervenche & & lys \\
arum      & iris & jacinthe & muguet
\end{NiceTabular}

```

rose	tulipe	marguerite	dahlia
violette	De très jolies fleurs		souci
pervenche			lys
arum	iris	jacinthe	muguet

⁷This value is the initial value of the *rounded corners* of Tikz.

⁸Tikz should be loaded (by default, `nicematrix` only loads PGF) and, if it's not, an error will be raised.

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

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

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

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 `\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 `draw` of the command `\Block` and to fill the background with rounded corners with the keys `fill` and `rounded-corners`.⁹
- It's possible to draw one or several borders of the cell with the key `borders`.

<code>\begin{NiceTabular}{cc}</code>	
<code>\toprule</code>	
<code>Writer & \Block[1]{year\\ of birth} \\</code>	Writer year
<code>\midrule</code>	of birth
<code>Hugo & 1802 \\</code>	Hugo 1802
<code>Balzac & 1799 \\</code>	Balzac 1799
<code>\bottomrule</code>	
<code>\end{NiceTabular}</code>	

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.¹⁰

⁹If one simply wishes to color the background of a unique cell, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl-like` is used).

¹⁰One 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 `!\qqquad` in the preamble which has been used to increase the space between the columns (this is not the behaviour of `\multicolumn`).

```
\begin{NiceTabular}{@{}c!\qqquadccc!\qqquadccc@{}}
\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 & 0.257 & 0.074 & 0.336 \\
\bottomrule
\end{NiceTabular}
```

Rank	First group			Second group		
	1A	1B	1C	2A	2B	2C
1	0.657	0.913	0.733	0.830	0.387	0.893
2	0.343	0.537	0.655	0.690	0.471	0.333
3	0.783	0.885	0.015	0.306	0.643	0.263
4	0.161	0.708	0.386	0.257	0.074	0.336

New 5.17 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\hline
Peter \\ \hline
Mary & George \\ \hline
\end{NiceTabular}
```

First	Second
Peter	
Mary	George

However, the vertical rules are not drawn in the blocks (created by `\Block`: cf. p. 4) nor in the corners (created by the key `corner`: cf. p. 10).

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

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

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
1	2	3	4
1	2	3	4

However, it's still possible to define a specifier (named, for instance, `I`) to draw vertical rules with the standard behaviour of `array`.

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

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 38):

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

5.1.2 The command `\cline`

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

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

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

A	B	C	D
A	<u>B</u>	C	D

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

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

A	<u>B</u>	C	D
A	<u>B</u>	C	D

5.2 The thickness and the color of the rules

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

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

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

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

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.

5.3 The tools of nicematrix for the rules

Here are the tools provided by `nicematrix` for the rules.

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

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

- These blocks are:
 - the blocks created by the command `\Block`¹¹ presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 19).
- The corners are created by the key `corners` explained below (see p. 10).

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

5.3.1 The keys `hlines` and `vlines`

The keys `hlines` and `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.

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

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

$$\left(\begin{array}{c|c|c|c|c|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{array} \right)$$

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}
```

rose	tulipe	marguerite	dahlia
violette	fleurs		souci
pervenche			lys
arum	iris	jacinthe	muguet

New 5.17 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.

¹¹And also the command `\multicolumn` but it’s recommended to use instead `\Block` in the environments of `nicematrix`.

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.¹²

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

In the example on the right (where B is in the center of a block of size 2×2), we have colored in blue the four (empty) corners of the array.

When the key **corners** is used, `nicematrix` computes the (empty) corners and these corners will be taken into account by the tools for drawing the rules (the rules won't be drawn in the corners).

Remark: In the previous versions of `nicematrix`, there was only a key `hvlines-except-corners` (now considered as obsolete).

```
\NiceMatrixOptions{cell-space-top-limit=3pt}
\begin{NiceTabular}{*{6}{c}}[corners,hvlines]
  & & & & A & \\
  & & A & A & A & \\
  & & & A & & \\
  & & A & A & A & A \\
  A & A & A & A & A & A \\
  A & A & A & A & A & A \\
  & A & A & A & & \\
  & \Block{2-2}{B} & & 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&4&6&4&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. 12).

¹²For 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.

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.¹³

```
\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}
```

$\begin{smallmatrix} y \\ x \end{smallmatrix}$	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

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

5.5 Dotted rules

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

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

$$\left(\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hdottedline 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{array} \right)$$

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

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

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

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`. Thus released, the letter “:” can be used otherwise (for example by the package `arydshln`¹⁴).

Remark: In the package `array` (on which the package `nicematrix` relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule¹⁵. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

6 The color of the rows and columns

6.1 Use of `colortbl`

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

Since the package `nicematrix` is based on `array`, it's possible to use `colortbl` with `nicematrix`.

However, there is two drawbacks:

¹³The author of this document considers that type of construction as graphically poor.

¹⁴However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

¹⁵In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

- 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 `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 `nicematrix` provides tools to avoid those problems.

6.2 The tools of `nicematrix` in the `\CodeBefore`

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁶

The extension `nicematrix` provides a key `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 `code-before` between the keywords `\CodeBefore` and `\Body` at the beginning of the environment.

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

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\chessboardcolors` and `arraycolor`.¹⁷

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 `corners` is used. This key has been described p. 10.

- The command `\cellcolor` takes its name from the command `\cellcolor` of `colortbl`.

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

¹⁶If you use Overleaf, Overleaf will do automatically the right number of compilations.

¹⁷Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-lj)” are also available to indicate the position to the potential rules: cf. p. 36.

```

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

```

a	b	c
e	f	g
h	i	j

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

```

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

```

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 17). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```

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

```

1	-1	1
-1	1	-1
1	-1	1

We have used the key `r` which aligns all the columns rightwards (cf. p. 31).

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

```

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

```

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}

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.
- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of `xcolor`¹⁸. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the tow 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 *i*- describes in fact the interval of all the rows of the tabular, beginning with the row *i*).

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

- The key `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 *i*-*j* (where *i* or *j* may be replaced by *).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.¹⁹
- With the key `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 \
Stephen & 8 \
Sarah & 18 \
Ashley & 20 \
Henry & 14 \
Madison & 15
\end{NiceTabular}
```

Results		
A	John	12
	Stephen	8
B	Sarah	18
	Ashley	20
	Henry	14
	Madison	15

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

John	12
	13
Steph	8
Sarah	18
	17
	15
Ashley	20
Henry	14
Madison	15
	19

¹⁸The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

¹⁹Otherwise, the color of a given row relies only upon the parity of its absolute number.

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 `corners` to require the determination of the corner *north east* (NE).

```
\begin{NiceTabular}{cccccc}[corners=NE,margin,hvlines,first-row,first-col]
\CodeBefore
  \rowcolors{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}
```

	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

One should remark that all the previous commands are compatible with the commands of `booktabs` (`\toprule`, `\midrule`, `\bottomrule`, etc). However, `booktabs` is not loaded by `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}{\rotate Price} \\
\cmidrule{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}
```

Product	dimensions (cm)			Price
	L	l	h	
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

We have used the type of column `S` of `siunitx`.

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.²⁰

There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

- `\cellcolor` which colorizes a cell;
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;
- `\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 `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble of the array).

²⁰Up to now, this key is *not* available in `\NiceMatrixOptions`.

```

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

```

Last name	First name	Birth day
Achard	Jacques	5 juin 1962
Lefebvre	Mathilde	23 mai 1988
Vanesse	Stephany	30 octobre 1994
Dupont	Chantal	15 janvier 1998

7 The command `\RowStyle`

New 5.18 The command `\RowStyle` takes in as argument some formatting intructions 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 available keys are `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` with the same meaning that the corresponding global keys (cf. p. 2).

```

\begin{NiceTabular}{cccc}[hlines,colortbl-like]
\RowStyle[cell-space-limits=3pt]{\rotate}
first & second & third & fourth \\
\rowcolor{blue!50}\RowStyle{\color{white}\sffamily}
1 & 2 & 3 & 4 \\
\end{NiceTabular}

```

first	second	third	fourth
1	2	3	4

The command `\rotate` is described p. 31.

8 The width of the columns

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

```

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

```

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns of an array directly with the key `columns-width`.

```

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

```

$$\begin{pmatrix} 1 & 12 & -123 \\ 12 & 0 & 0 \\ 4 & 1 & 2 \end{pmatrix}$$

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

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.²¹

```


$$\begin{pmatrix} 1 & 12 & -123 \\ 12 & 0 & 0 \\ 4 & 1 & 2 \end{pmatrix}$$


```

Without surprise, it's possible to fix the minimal width of the columns of all the matrices of a current scope with the command `\NiceMatrixOptions`.

```


$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 1245 \\ 345 & 2 \end{pmatrix}$$


```

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`²². The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

```


$$\begin{bmatrix} 9 & 17 \\ -2 & 5 \end{bmatrix} \quad \begin{bmatrix} 1 & 1245345 \\ 345 & 2 \end{bmatrix}$$


```

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`.

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

```


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


```

²¹The result is achieved with only one compilation (but PGF/Tikz will have written informations in the `.aux` file and a message requiring a second compilation will appear).

²²At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

```

& C_1 & \Cdots & & C_4 & \\
\end{pNiceMatrix}$

```

$$\begin{array}{c}
C_1 \dots\dots\dots C_4 \\
L_1 \left(\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) L_1 \\
\vdots \\
\vdots \\
L_4 \left(\begin{array}{cccc} a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) L_4 \\
C_1 \dots\dots\dots C_4
\end{array}$$

The dotted lines have been drawn with the tools presented p. 19.

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 `r` for the first column and `l` for the last one.²³
- 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 `light-syntax` (cf. p. 33) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

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

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

```

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

```

²³The users wishing exteriors columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 25).

$$\begin{array}{c}
\textcolor{red}{C_1} \cdots \cdots \cdots \textcolor{red}{C_4} \\
\textcolor{blue}{L_1} \left(\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) \textcolor{violet}{L_4} \\
\textcolor{green}{C_1} \cdots \cdots \cdots \textcolor{green}{C_4}
\end{array}$$

Remarks

- As shown in the previous example, the horizontal and vertical rules doesn't extend in the exterior rows and columns.
However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 38.
- A specification of color present in `code-for-first-row` also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 16) doesn't apply to the “first column” and “last column”.
- For technical reasons, it's not possible to use the option of the command `\` after the “first row” or before the “last row”. The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 25.

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`.²⁴

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells²⁵ 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`.²⁶

```

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

```

$$\left[\begin{array}{ccccccc}
a_1 & \cdots & \cdots & \cdots & \cdots & \cdots & a_1 \\
\vdots & & & & & & \\
\vdots & & a_2 & \cdots & \cdots & \cdots & a_2 \\
\vdots & & \vdots & & & & \\
\vdots & & \vdots & & & & \\
\vdots & & \vdots & & & & \\
a_1 & a_2 & & & & & a_n
\end{array} \right]$$

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}

```

$$\left[\begin{array}{ccccccc}
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & & & & & \\
\vdots & & & & & & \\
\vdots & & & & & & \\
\vdots & & & & & & \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0
\end{array} \right]$$

²⁴The 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`.

²⁵The precise definition of a “non-empty cell” is given below (cf. p. 39).

²⁶It's also possible to change the color of all theses dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 23.

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

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

$$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$$

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

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

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

$$\begin{bmatrix} 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ & & & \vdots \\ 0 & & \cdots & 0 \end{bmatrix}$$

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\Vdots` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.²⁷

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

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

$$\begin{bmatrix} 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & & 0 \end{bmatrix}$$

10.1 The option `nullify-dots`

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

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

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

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

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

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

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

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

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

²⁷In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 16

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

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

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

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

10.2 The commands `\Hdotsfor` and `\Vdotsfor`

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

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

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

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

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

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

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

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`²⁸ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

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

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

²⁸We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

$$\left[\begin{array}{cccc} C[a_1, a_1] & \cdots & C[a_1, a_n] & \cdots & C[a_1, a_1^{(p)}] & \cdots & C[a_1, a_n^{(p)}] \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ C[a_n, a_1] & \cdots & C[a_n, a_n] & \cdots & C[a_n, a_1^{(p)}] & \cdots & C[a_n, a_n^{(p)}] \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ C[a_1^{(p)}, a_1] & \cdots & C[a_1^{(p)}, a_n] & \cdots & C[a_1^{(p)}, a_1^{(p)}] & \cdots & C[a_1^{(p)}, a_n^{(p)}] \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ C[a_n^{(p)}, a_1] & \cdots & C[a_n^{(p)}, a_n] & \cdots & C[a_n^{(p)}, a_1^{(p)}] & \cdots & C[a_n^{(p)}, a_n^{(p)}] \end{array} \right]$$

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 `nicematrix` without having to modify the code of each matrix. It's possible with the keys `renew-dots` and `renew-matrix`.²⁹

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`²⁴ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

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

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

$$\begin{pmatrix} 1 & \cdots & \cdots & 1 & \\ 0 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \vdots & \\ 0 & \cdots & 0 & & 1 \end{pmatrix}$$

10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 24) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

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

$$\begin{bmatrix} 1 & & & 0 \\ & \ddots^{n \text{ times}} & & \\ 0 & & & 1 \end{bmatrix}$$

²⁹The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 24) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 17.

The option `xdots/shorten`

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

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).³⁰

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

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

However (when Tikz is loaded) it’s possible to use for `xdots/line-style` any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tikz pathes (with the exception of “`color`”, “`shorten >`” and “`shorten <`”).

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

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

$$\begin{pmatrix} a & b & 0 & \cdots & 0 \\ b & a & b & \cdots & \\ 0 & b & a & \cdots & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & b & a \end{pmatrix}$$

³⁰The 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.

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` and by the keys `hlines`, `vlines` and `hvlines` are not drawn within the blocks).³¹

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

$$\left[\begin{array}{ccc|c} & & & 0 \\ & & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array} \right]$$

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³²

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 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. 34.

Moreover, two special commands are available in the `\CodeAfter`: `\line` and `\SubMatrix`.

11.1 The command `\line` in the `\CodeAfter`

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form *i-j* where *i* is the number of the row and *j* is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 23).

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

```
\NiceMatrixOptions{xdots/shorten = 0.6 em}
\begin{pNiceMatrix}
I & 0 & & \Cdots & 0 & \\
0 & & I & & \Ddots & \Vdots \\
\Vdots & & \Ddots & & I & 0 \\
0 & & \Cdots & & 0 & I
\CodeAfter \line{2-2}{3-3}
\end{pNiceMatrix}
```

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

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. 39).

³¹On the other side, the command `\line` in the `\CodeAfter` (cf. p. 24) does *not* create block.

³²There is also a key `code-before` described p. 12.


```

\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      & \\
\CodeAfter \line[shorten=6pt]{1-5}{4-7}
\end{bNiceMatrix}

```

$$\left[\begin{array}{cc|cc} 1 & \cdots & 1 & 2 \\ 0 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 \end{array} \right]$$

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.³³

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.

```

\[\begin{NiceArray}{ccc@{\hspace{1.5em}}c}[cell-space-limits=2pt,margin]
1      & & 1      & & 1      & & x \\\
\dfrac{1}{4} & & \dfrac{1}{2} & & \dfrac{1}{4} & & y \\\
1      & & 2      & & 3      & & z \\
\CodeAfter
  \SubMatrix({1-1}{3-3})
  \SubMatrix({1-4}{3-4})
\end{NiceArray}\]

```

$$\begin{pmatrix} 1 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

New 5.18 In fact, the command `\SubMatrix` also takes in two optional arguments specified by the traditional symbols `^` and `_` for material in superscript and subscript.

```

$\begin{bNiceMatrix}[right-margin=1em]
1 & 1 & 1 \\\
1 & a & b \\\
1 & c & d \\
\CodeAfter
  \SubMatrix[{2-2}{3-3}]^T
\end{bNiceMatrix}$

```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & b \\ 1 & c & d \end{bmatrix}^T$$

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

³³There 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}]`).

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

```


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


```

Here is the same example with the key `slim` used for one of the submatrices.

```


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


```

There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 37.

It's also possible to specify some delimiters³⁴ 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 `\enskip` is automatically inserted.

³⁴Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

```

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

```

$$\begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} \begin{pmatrix} a_{12} \\ \int_0^1 \frac{1}{x^2+1} dx \\ a_{32} \end{pmatrix} \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}$$

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). 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 \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 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

```

Last name	First name	Birth day
Achard ^a	Jacques	June 5, 2005
Lefebvre ^b	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

^a Achard is an old family of the Poitou.

^b The name Lefebvre is an alteration of the name Lefebure.

- If you have several successive commands `\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 `multiple` of `footmisc` for the footnotes).
- If a command `\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 `notes/para` is used, the notes are composed at the end of the array in a single paragraph (as with the key `para` of `threeparttable`).
- There is a key `tabularnote` which provides a way to insert some text in the zone of the notes before the numbered tabular notes.
- If the package `booktabs` has been loaded (before or after `nicematrix`), the key `notes/bottomrule` draws a `\bottomrule` of `booktabs` after the notes.
- The command `\tabularnote` may be used before the environment of `nicematrix`. Thus, it's possible to use it on the title inserted by `\caption` in an environment `{table}` of LaTeX.
- It's possible to create a reference to a tabular note created by `\tabularnote` (with the usual command `\label` used after the `\tabularnote`).

For an illustration of some of those remarks, see table 1, p. 29. This table has been composed with the following code.

```
\begin{table}
\setlength{\belowcaptionskip}{1ex}
\centering
\caption{Use of \texttt{\textbackslash tabularnote}\tabularnote{It's possible
to put a note in the caption.}}
\label{t:tabularnote}
\begin{NiceTabular}{@{}llc@{}}
[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.}\tabularnote{Nicknamed ``the Lady with the Lamp''.}
& Florence & 90 \\
Schoelcher & Victor & 89\tabularnote{The label of the note is overlapping.}\\
Touchet & Marie & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}
```

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

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

We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.

Initial value: `false`

That key is also available within a given environment.

Table 1: Use of `\tablarnote`^a

Last name	First name	Length of life
Churchill	Wiston	91
Nightingale ^{b,c}	Florence	90
Schoelcher	Victor	89 ^d
Touchet	Marie	89
Wallis	John	87

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 The label of the note is overlapping.

- 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: `#1`

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

```
\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: `#1`

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

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

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

- The notes are composed at the end of the tabular by using internally a style of list of `enumitem`. The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that type of list.

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

This initial value contains the specification `align = left` which requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 29).

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

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

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

Initial value: `empty`

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

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

It's also possible to add `\raggedright` or `\RaggedRight` in that key (`\RaggedRight` is a command of `ragged2e`).

For an example of customisation of the tabular notes, see p. 41.

12.4 Use of `{NiceTabular}` with `threeparttable`

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

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

13 Other features

13.1 Use of the column type `S` of `siunitx`

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

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

$$\begin{pmatrix} C_1 & \dots & C_n \\ 2.3 & 0 & \dots & 0 \\ 12.4 & \vdots & & \vdots \\ 1.45 & \vdots & & \vdots \\ 7.2 & 0 & \dots & 0 \end{pmatrix}$$

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

13.2 Alignment option in `{NiceMatrix}`

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

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

$$\begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix}$$

13.3 The command `\rotate`

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

In the following command, we use that command in the `code-for-first-row`.³⁵

³⁵It can also be used in `RowStyle` (cf. p. 16).

```

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

```

$$A = \begin{pmatrix} \text{image of } e_1 & \text{image of } e_2 & \text{image of } e_3 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix}$$

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

```

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

```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix}$$

13.4 The option `small`

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

```

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

```

$$\left[\begin{array}{cccc|c} 1 & -2 & 3 & 4 & 5 \\ 0 & 3 & 2 & 1 & 2 \\ 0 & 1 & 1 & 2 & 3 \end{array} \right] \begin{matrix} \\ L_2 \leftarrow 2L_1 - L_2 \\ L_3 \leftarrow L_1 + L_2 \end{matrix}$$

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

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

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

13.5 The counters iRow and jCol

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column³⁶. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 12) and in the `\CodeAfter` (cf. p. 24), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```
$\begin{pNiceMatrix}% don't forget the %
  [first-row,
   first-col,
   code-for-first-row = \mathbf{\alphajCol} ,
   code-for-first-col = \mathbf{\arabic{iRow}} ]
& & & & \\
& 1 & 2 & 3 & 4 \\
& 5 & 6 & 7 & 8 \\
& 9 & 10 & 11 & 12
\end{pNiceMatrix}$
```

$$\begin{matrix} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ \mathbf{1} & \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \\ \mathbf{2} & \begin{pmatrix} 5 & 6 & 7 & 8 \end{pmatrix} \\ \mathbf{3} & \begin{pmatrix} 9 & 10 & 11 & 12 \end{pmatrix} \end{matrix}$$

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

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

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

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

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

13.6 The option light-syntax

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

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

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

$$\begin{matrix} & a & & b \\ a & \begin{bmatrix} 2 \cos a & \cos a + \cos b \end{bmatrix} \\ b & \begin{bmatrix} \cos a + \cos b & 2 \cos b \end{bmatrix} \end{matrix}$$

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.³⁷

³⁶We 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.

³⁷The 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.

13.7 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`.

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

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 25).

13.8 The environment `{NiceArrayWithDelims}`

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

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

$$\begin{array}{ccc} \downarrow & 1 & 2 & 3 & \uparrow \\ & 4 & 5 & 6 & \\ & 7 & 8 & 9 & \end{array}$$

14 Use of Tikz with nicematrix

14.1 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`. ³⁸

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

In the environment with the number n , the node of the row i and column j has for name `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) ;
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & \textcircled{5} & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

³⁸One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 19) and the computation of the “corners” (cf. p. 10).

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) ;
 $\end{pNiceMatrix}$ 

```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & \textcircled{5} & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 47).

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

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

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.³⁹

These nodes are not used by `nicematrix` by default, and that's why they are not created by default.

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

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.⁴⁰

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

The “large nodes” of the first column and last column may appear too small for some usage. That's why it's possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.⁴¹

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

³⁹There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

⁴⁰There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 17).

⁴¹The 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.

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.

$$\left(\begin{array}{|c|c|c|} \hline a & a+b & a+b+c \\ \hline a & a & a+b \\ \hline a & a & a \\ \hline \end{array} \right)$$

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:

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

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

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

The nodes we have described are not available by default in the `\CodeBefore` (described p. 12). 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).

14.3 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`.

$\bullet^{1.5}$	tulipe	lys
arum	$\bullet^{2.5}$	violette mauve
muguet	dahlia	$\bullet^{3.5}$

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 $(i-j)$.

```

\begin{NiceMatrix}
\CodeBefore
\tikz \draw [fill=red!15] (7-|4) |- (8-|5) |- (9-|6) |- cycle ;
\Body
1 \\\
1 & 1 \\\
1 & 2 & 1 \\\
1 & 3 & 3 & 1 \\\
1 & 4 & 6 & 4 & 1 \\\
1 & 5 & 10 & 10 & 5 & 1 \\\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\\
1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1
\end{NiceMatrix}

```

```

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

```

The nodes of the form $i.5$ may be used, for example to cross a row of a matrix (if Tikz is loaded).

```

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

```

$$\begin{pmatrix} 2 & 1 & 3 & | & 0 \\ 3 & 3 & 1 & | & 0 \\ \hline 3 & 3 & 1 & | & 0 \end{pmatrix}$$

14.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 25.

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

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

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

$$\begin{pmatrix} 121 & 23 & 345 & 345 \\ 45 & \left\{ \begin{array}{cc} 346 & 863 \\ 38458 & 34 \end{array} \right\} & 444 \\ 3462 & & 294 \\ 34 & 7 & 78 & 309 \end{pmatrix}$$

15 API for the developers

The package `nicematrix` provides two variables which are internal but public⁴²:

- `\g_nicematrix_code_before_tl` ;
- `\g_nicematrix_code_after_tl`.

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

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

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

```
ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_hatch:nnn
{
  \tikz \fill [ pattern = north-west-lines , pattern-color = #3 ]
    ( #1 -| #2 ) rectangle ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;
}

\NewDocumentCommand \hatchcell { ! 0 { black } }
{
  \tl_gput_right:Nx \g_nicematrix_code_before_tl
    { \__pantigny_hatch:nnn { \arabic { iRow } } { \arabic { jCol } } { #1 } }
}
\ExplSyntaxOff
```

Here is an example of use:

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

Tokyo	Paris	London
Lima	Oslo	Miami
Los Angeles	Madrid	Roma

16 Technical remarks

16.1 Definition of new column types

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

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job⁴³:

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

⁴²According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

⁴³The command `\vrule` is a TeX (and not LaTeX) command.

The heavy vertical rule won't extend in the exterior rows.⁴⁴

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

$$\begin{pmatrix} C_1 & C_2 & C_3 & C_4 \\ a & b & c & d \\ e & f & g & h \\ C_1 & C_2 & C_3 & C_4 \end{pmatrix}$$

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

16.2 Diagonal lines

By default, all the diagonal lines⁴⁵ of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

$$A = \begin{pmatrix} 1 & \cdots & \cdots & \cdots & 1 \\ a+b & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ a+b & \cdots & \cdots & a+b & 1 \end{pmatrix}$$

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

$$A = \begin{pmatrix} 1 & \cdots & \cdots & \cdots & 1 \\ a+b & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ a+b & \cdots & \cdots & a+b & 1 \end{pmatrix}$$

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

The same example without parallelization:

$$A = \begin{pmatrix} 1 & \cdots & \cdots & \cdots & 1 \\ a+b & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ a+b & \cdots & \cdots & a+b & 1 \end{pmatrix}$$

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[draw-first]`.

16.3 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. However, an “empty cell” is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands `&`). The precise rules are as follow.

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

⁴⁴Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won’t cross the double rules of `\hline\hline`.

⁴⁵We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

```

\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. *Caution* : One should not rely upon that point because it may change if a future version of `nicematrix`.

16.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁴⁶. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁴⁷. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

16.5 Incompatibilities

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

Anyway, in order to use `arydshln`, one must first free the letter “:” by giving a new letter for the vertical dotted rules of `nicematrix`:

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

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

the package `nicematrix` is not compatible with the class `ieeeaccess` (because that class is not compatible with PGF/Tikz).

⁴⁶In the documentation of `{amsmath}`, we can read: *The extra space of `\arraycolsep` that `array` adds on each side is a waste so we remove it [in `{matrix}`] (perhaps we should instead remove it from `array` in general, but that's a harder task).*

⁴⁷And not by inserting `@{}` on both sides of the preamble of the array. As a consequence, the length of the `\hline` is not modified and may appear too long, in particular when using square brackets.

17 Examples

17.1 Utilisation of the key “tikz” of the command \Block

```
\ttfamily \small
\begin{NiceTabular}{m{4.5cm}m{4.5cm}m{4.5cm}}[hvlines]
  \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}
& \Block[tikz={left color = blue!50}]{}
  {left color = blue!50} \,\,
\end{NiceTabular}
```

<pre>pattern = grid, pattern color = lightgray</pre>	<pre>pattern = north west lines, pattern color = blue</pre>	<pre>outer color = red!50, inner color = white</pre>
<pre>pattern = sixpointed stars, pattern color = blue!15</pre>	<pre>left color = blue!50</pre>	

17.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 27.

Let’s consider that we wish to number the notes of a tabular with stars.⁴⁸

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument⁴⁹

```
\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 =
    {
```

⁴⁸Of course, it’s realistic only when there is very few notes in the tabular.

⁴⁹In fact: the value of its argument.

```

        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 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

```

Last name	First name	Birth day
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.

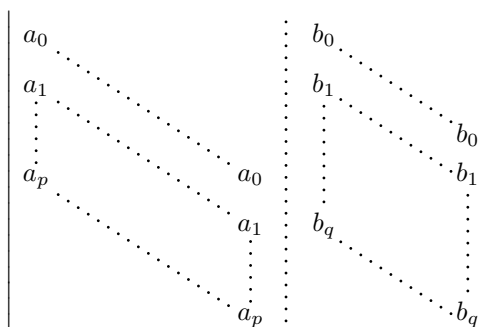
17.3 Dotted lines

An example with the resultant of two polynoms:

```

\setlength{\extrarowheight}{1mm}
\begin{vNiceArray}{ccccccc}[columns-width=6mm]
a_0 & & & & b_0 & & \\
a_1 & & \Ddots & & b_1 & & \Ddots \\
\vdots & & \Ddots & & \vdots & & \Ddots b_0 \\
a_p & & \Ddots a_0 & & & & b_1 \\
& & \Ddots a_1 & & b_q & & \Ddots \\
& & \Ddots \vdots & & & & \Ddots \\
& & \Ddots a_p & & & & b_q \\
\end{vNiceArray}

```



An example for a linear system:

```

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

```

$$\left(\begin{array}{cccccc|c} 1 & 1 & 1 & \dots & 1 & 0 \\ 0 & 1 & 0 & \dots & 0 & \vdots \\ 0 & 0 & 1 & \dots & 0 & \vdots \\ \vdots & & & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & 1 & 0 \end{array} \right) \begin{array}{l} L_2 \leftarrow L_2 - L_1 \\ L_3 \leftarrow L_3 - L_1 \\ \vdots \\ L_n \leftarrow L_n - L_1 \end{array}$$

17.4 Dotted lines which are no longer dotted

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

```

\NiceMatrixOptions{code-for-first-row = \scriptstyle,code-for-first-col = \scriptstyle }
\setcounter{MaxMatrixCols}{12}
\newcommand{\blue}{\color{blue}}
\begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
1&&&\Vdots&&&\Vdots\\
&\Ddots[line-style=standard]\\
&&1\\
\Cdots[color=blue,line-style=dashed]&&&\blue 0&
\Cdots&&\blue 1&&&\Cdots&\blue \leftarrow i\\
&&&1\\
&&\Vdots&&\Ddots[line-style=standard]&&\Vdots\\
&&&&&1\\
\Cdots&&\blue 1&\Cdots&&\Cdots&\blue 0&&\Cdots&\blue \leftarrow j\\
&&&&&1\\
&&&&&&\Ddots[line-style=standard]\\
&&\Vdots&&&\Vdots&&1\\
&&\blue \overset{\uparrow}{i}&&&\blue \overset{\uparrow}{j}
\end{pNiceMatrix}

```

$$\left(\begin{array}{cccccc|c} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ \hline & & & 0 & & 1 & \leftarrow i \\ & & & & \ddots & & \\ & & & & & 1 & \\ \hline & & & 1 & & 0 & \leftarrow j \\ & & & & & & \ddots \\ & & & & & & 1 \end{array} \right)$$

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.

```

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

```

$$\begin{array}{c}
 \xrightarrow{n \text{ columns}} \\
 \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} \\
 \uparrow n \text{ rows}
 \end{array}$$

17.5 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 `\NiceMatrixBlock` and its option `auto-columns-width`.

```

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
  {
    light-syntax,
    last-col, code-for-last-col = \color{blue} \scriptstyle,
  }
\setlength{\extrarowheight}{1mm}

$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
3 -18 12 1 4 ;
-3 -46 29 -2 -15 ;
9 10 -5 4 7
\end{pNiceArray}$

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

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

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
0 64 -41 1 19 ;

```

```
\end{pNiceArray}$
```

```
\end{NiceMatrixBlock}
```

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{pmatrix}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{pmatrix} \begin{matrix} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{matrix}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} L_3 \leftarrow 3L_2 + L_3$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{pmatrix}$$

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 slimer).

In order to solve that problem, it's possible to require the delimiters to be composed with the maximal width, thanks to the boolean key `delimiters/max-width`.

```
\begin{NiceMatrixBlock}[auto-columns-width]
```

```
\NiceMatrixOptions
```

```
{
```

```
delimiters/max-width,
```

```
light-syntax,
```

```
last-col, code-for-last-col = \color{blue}\scriptstyle,
```

```
}
```

```
\setlength{\extrarowheight}{1mm}
```

```
$$\begin{pNiceArray}{rrrr|r}
```

```
12 -8 7 5 3 {} ;
```

```
3 -18 12 1 4 ;
```

```
-3 -46 29 -2 -15 ;
```

```
9 10 -5 4 7
```

```
\end{pNiceArray}$$
```

```
...
```

```
\end{NiceMatrixBlock}
```

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{pmatrix}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{pmatrix} \begin{matrix} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{matrix}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)_{L_3 \leftarrow 3L_2 + L_3}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{array}\right)$$

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}
\[\begin{NiceMatrix}[ r, last-col=6, code-for-last-col = \scriptstyle \color{blue} ]
12 & -8 & 7 & 5 & 3 & \\
3 & -18 & 12 & 1 & 4 & \\
-3 & -46 & 29 & -2 & -15 & \\
9 & 10 & -5 & 4 & 7 & \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & L_2 \gets L_1 - 4L_2 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1 + 4L_3 \\
0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1 - 4L_4 \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2 + L_3 \\[1mm]
12 & -8 & 7 & 5 & 3 & \\
0 & 64 & -41 & 1 & 19 & \\
\CodeAfter [sub-matrix/vlines=4]
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceMatrix}\]
```

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{array}\right)$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{array}\right)_{\begin{array}{l} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{array}}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)_{L_3 \leftarrow 3L_2 + L_3}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{array}\right)$$

17.6 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 `draw` of the command `\Block` (this is one of the uses of a mono-cell block⁵⁰).

```
\begin{pNiceArray}{>{\strut}cccc}[margin,rules/color=blue]
\Block[draw]{a_{11}} & a_{12} & a_{13} & a_{14} \\
a_{21} & \Block[draw]{a_{22}} & a_{23} & a_{24} \\
a_{31} & a_{32} & \Block[draw]{a_{33}} & a_{34} \\
a_{41} & a_{42} & a_{43} & \Block[draw]{a_{44}} \\
\end{pNiceArray}
```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines` and `hvlines` spread the cells.⁵¹

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

```
\begin{pNiceArray}{>{\strut}cccc}[margin, extra-margin=2pt,colortbl-like]
\rowcolor{red!15}A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & \rowcolor{red!15}A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & \rowcolor{red!15}A_{33} & A_{34} \\
A_{41} & A_{42} & A_{43} & \rowcolor{red!15}A_{44} \\
\end{pNiceArray}
```

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix}$$

However, it's not possible to do a fine tuning. That's why we describe now a method to highlight a row of the matrix. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend mode` equal to `multiply`.

Caution : Some PDF readers are not able to show transparency.⁵²

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

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

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 `\CodeBefore` (for efficiency). We have to require their creation with the key `create-cell-nodes` of the keyword `\CodeBefore`.

⁵⁰We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁵¹For the command `\cline`, see the remark p. 8.

⁵²In Overleaf, the “built-in” PDF viewer does not show transparency. You can switch to the “native” viewer in that case.

```

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

$\begin{bNiceMatrix}
\CodeBefore [create-cell-nodes]
\tikz \node [highlight = (2-1) (2-3)] {} ;
\Body
0 & \Cdots & 0 \\
1 & \Cdots & 1 \\
0 & \Cdots & 0 \\
\end{bNiceMatrix}$

```

$$\begin{bmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 1 \\ 0 & \cdots & 0 \end{bmatrix}$$

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

```

\[\begin{pNiceArray}{ccc}[last-col]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture}
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray}\]

```

$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

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

```

\[\begin{pNiceArray}{ccc}[last-col,create-medium-nodes]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture} [name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\end{pNiceArray}\]

```

$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

17.7 Utilisation of `\SubMatrix` in the `\CodeBefore`

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

The whole figure is an environment `{NiceArray}` and the three pairs of parenthesis have been added with `\SubMatrix` in the `\CodeBefore`.

$$L_i \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{i1} & \cdots & a_{in} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1n} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & b_{nn} \end{pmatrix} \begin{pmatrix} \vdots \\ \vdots \\ c_{ij} \\ \vdots \end{pmatrix}$$

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

[ \begin{NiceArray}{*{6}{c}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-11})
  \SubMatrix({7-2}{11-6})
  \SubMatrix({7-7}{11-11})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
  & & & & & & & \color{blue}\scriptstyle C_j \\\
  & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\\
  & & & & & \Vdots & & \Vdots & & \Vdots \\\
  & & & & & & & b_{kj} \\\
  & & & & & & & \Vdots \\\
  & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\\[3mm]
  & a_{11} & \Cdots & & & a_{1n} \\\
  & \Vdots & & & & \Vdots & & & & \Vdots \\\
\color{blue}\scriptstyle L_i
  & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots & & c_{ij} \\\
  & \Vdots & & & & \Vdots \\\
  & a_{n1} & \Cdots & & & a_{nn} \\\
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\]
```

18 Implementation

By default, the package `nicematrix` doesn't patch any existing code.

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\ddots` and `\iddots` are redefined in the environments provided by `nicematrix` as explained previously. In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `amsmath` is redefined.

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

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

Declaration of the package and packages loaded

The prefix `nicematrix` has been registered for this package.

See: <http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf>

<@@=nicematrix>

First, we load `pgfcore` and the module `shapes`. We do so because it's not possible to use `\usepgfmodule` in `\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 `\usepackage` is at the end of this package for technical reasons.

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

```
9 \RequirePackage { xparse }
10 \RequirePackage { array }
11 \RequirePackage { amsmath }

12 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
13 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
14 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
15 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
17 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
18 \cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }

19 \cs_new_protected:Npn \@@_msg_redirect_name:nn
20   { \msg_redirect_name:nnn { nicematrix } }
```

Technical definitions

```
21 \bool_new:N \c_@@_in_preamble_bool
22 \bool_set_true:N \c_@@_in_preamble_bool
23 \AtBeginDocument { \bool_set_false:N \c_@@_in_preamble_bool }

24 \bool_new:N \c_@@_arydshln_loaded_bool
25 \bool_new:N \c_@@_booktabs_loaded_bool
26 \bool_new:N \c_@@_enumitem_loaded_bool
27 \bool_new:N \c_@@_tikz_loaded_bool
```

```

28 \AtBeginDocument
29 {
30   \@ifpackageloaded { arydshln }
31     { \bool_set_true:N \c_@@_arydshln_loaded_bool }
32     { }
33   \@ifpackageloaded { booktabs }
34     { \bool_set_true:N \c_@@_booktabs_loaded_bool }
35     { }
36   \@ifpackageloaded { enumitem }
37     { \bool_set_true:N \c_@@_enumitem_loaded_bool }
38     { }
39   \@ifpackageloaded { tikz }
40     {

```

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

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

```

41     \bool_set_true:N \c_@@_tikz_loaded_bool
42     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
43     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
44   }
45   {
46     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
47     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
48   }
49 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date January 2021, the current version `revtex4-2` is 4.2e (compatible with `booktabs`).

```

50 \bool_new:N \c_@@_revtex_bool
51 \@ifclassloaded { revtex4-1 }
52   { \bool_set_true:N \c_@@_revtex_bool }
53   { }
54 \@ifclassloaded { revtex4-2 }
55   { \bool_set_true:N \c_@@_revtex_bool }
56   { }

```

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

```

57 \cs_if_exist:NT \rvtx@ifformat@geq { \bool_set_true:N \c_@@_revtex_bool }

58 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

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

```

59 \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.

```

60 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
61 {
62   \iow_now:Nn \@mainaux
63   {
64     \ExplSyntaxOn
65     \cs_if_free:NT \pgfsyspdfmark
66       { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
67     \ExplSyntaxOff

```

```

68     }
69     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
70 }

```

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

```

71 \ProvideDocumentCommand \iddots { }
72 {
73     \mathinner
74     {
75         \tex_mkern:D 1 mu
76         \box_move_up:nn { 1 pt } { \hbox:n { . } }
77         \tex_mkern:D 2 mu
78         \box_move_up:nn { 4 pt } { \hbox:n { . } }
79         \tex_mkern:D 2 mu
80         \box_move_up:nn { 7 pt }
81         { \vbox:n { \kern 7 pt \hbox:n { . } } }
82         \tex_mkern:D 1 mu
83     }
84 }

```

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.

```

85 \AtBeginDocument
86 {
87     \@ifpackageloaded { booktabs }
88     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
89     { }
90 }
91 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
92 {
93     \cs_set_eq:NN \@@_old_pgful@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`).

```

94     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
95     {
96         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
97         { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
98     }
99 }

```

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

```

100 \bool_new:N \c_@@_colortbl_loaded_bool
101 \AtBeginDocument
102 {
103     \@ifpackageloaded { colortbl }
104     {
105         \bool_set_true:N \c_@@_colortbl_loaded_bool
106     }
107     {

```

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.

```

108     \cs_set_protected:Npn \CT@arc@ { }
109     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
110     \cs_set:Npn \CT@arc #1 #2
111     {

```

```

112         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
113         { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
114     }

```

Idem for \CT@drs@.

```

115     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
116     \cs_set:Npn \CT@drs #1 #2
117     {
118         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
119         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
120     }
121     \cs_set:Npn \hline
122     {
123         \noalign { \ifnum 0 = ` } \fi
124         \cs_set_eq:NN \hskip \vskip
125         \cs_set_eq:NN \vrule \hrule
126         \cs_set_eq:NN \@width \@height
127         { \CT@arc@ \vline }
128         \futurelet \reserved@a
129         \@xhline
130     }
131 }
132 }

```

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.

```

133 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
134 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
135 {
136     \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
137     \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
138     \multispan { \int_eval:n { #2 - #1 + 1 } }
139     {
140         \CT@arc@
141         \leaders \hrule \@height \arrayrulewidth \hfill

```

The following \skip_horizontal:N \c_zero_dim is to prevent a potential \unskip to delete the \leaders⁵³

```

142     \skip_horizontal:N \c_zero_dim
143 }

```

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

```

144     \everycr { }
145     \cr
146     \noalign { \skip_vertical:N -\arrayrulewidth }
147 }

```

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.

```

148 \cs_set:Npn \@@_cline

```

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

```

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

```

The command \cline_i:nn has two arguments. The first is the number of the current column (it *must* be used in that column). The second is a standard argument of \cline of the form *i-j*.

```

150 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1-#2 \q_stop }
151 \cs_set:Npn \@@_cline_i:w #1-#2-#3 \q_stop
152 {

```

⁵³See question 99041 on TeX StackExchange.

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).

```

153 \int_compare:nNnT { #1 } < { #2 }
154   { \multispan { \int_eval:n { #2 - #1 } } & }
155 \multispan { \int_eval:n { #3 - #2 + 1 } }
156   {
157     \CT@arc@
158     \leaders \hrule \@height \arrayrulewidth \hfill
159     \skip_horizontal:N \c_zero_dim
160   }

```

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

```

161 \peek_meaning_remove_ignore_spaces:NTF \cline
162   { & \@@_cline_i:en { \@@_succ:n { #3 } } }
163   { \everycr { } \cr }
164 }
165 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following commands are only for efficiency. They must *not* be protected because it will be used (for instance) in names of PGF nodes.

```

166 \cs_new:Npn \@@_succ:n #1 { \the \numexpr #1 + 1 \relax }
167 \cs_new:Npn \@@_pred:n #1 { \the \numexpr #1 - 1 \relax }

```

The following command is a small shortcut.

```

168 \cs_new:Npn \@@_math_toggle_token:
169   { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

170 \cs_new_protected:Npn \@@_set_CT@arc@:
171   { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
172 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
173   { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
174 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
175   { \cs_set:Npn \CT@arc@ { \color { #1 } } }

176 \cs_set_eq:NN \@@_old_pgfpintanchor \pgfpintanchor

```

The column S of siunitx

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

```

177 \bool_new:N \c_@@_siunitx_loaded_bool
178 \AtBeginDocument
179   {
180     \ifpackageloaded { siunitx }
181       { \bool_set_true:N \c_@@_siunitx_loaded_bool }
182       { }
183   }

```

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

```

184 \AtBeginDocument
185   {
186     \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
187     { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
188     {

```

For version of siunitx at least equal to 3.0, the adaptation is different from previous ones. We test the version of siunitx by the existence of the control sequence \siunitx_cell_begin:w.

```

189     \cs_if_exist:NTF \siunitx_cell_begin:w
190     {
191       \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
192       {

```

```

193         \renewcommand*{\NC@rewrite@S}[1] []
194         {
195             \@temptokena \exp_after:wN
196             {
197                 \tex_the:D \@temptokena
198                 > {
199                     \@@_Cell:
200                     \keys_set:nn { siunitx } { ##1 }
201                     \siunitx_cell_begin:w
202                 }
203     \@@_true_c: will be replaced statically by c at the end of the construction of the preamble.
204     \@@_true_c:
205     < { \siunitx_cell_end: \@@_end_Cell: }
206     }
207     \NC@find
208     }
209     }
210     {
211         \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
212         {
213             \renewcommand*{\NC@rewrite@S}[1] []
214             {
215                 \@temptokena \exp_after:wN
216                 {
217                     \tex_the:D \@temptokena
218                     > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
219                     \@@_true_c:
220                     < { \c_@@_table_print_tl \@@_end_Cell: }
221                 }
222                 \NC@find
223             }
224         }
225     }
226 }
227 }

```

The following code is used to define `\c_@@_table_collect_begin_tl` and `\c_@@_table_print_tl` when the version of `siunitx` is prior to 3.0. The command `\@@_adapt_S_column` is used in the environment `{NiceArrayWithDelims}`.

```

228 \AtBeginDocument
229 {
230     \cs_set_eq:NN \@@_adapt_S_column: \prg_do_nothing:
231     \bool_lazy_and:nnT
232     { \c_@@_siunitx_loaded_bool }
233     { ! \cs_if_exist_p:N \siunitx_cell_begin:w }
234     {
235         \cs_set_protected:Npn \@@_adapt_S_column:
236         {
237             \group_begin:
238             \@temptokena = { }
239             \cs_set_eq:NN \NC@find \prg_do_nothing:
240             \NC@rewrite@S { }
241             \tl_gset:NV \g_tmpa_tl \@temptokena
242             \group_end:
243             \tl_new:N \c_@@_table_collect_begin_tl
244             \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
245             \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
246             \tl_new:N \c_@@_table_print_tl
247             \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
248             \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
249         }
250     }

```

251 }

Parameters

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

```
252 \bool_new:N \c_@@_define_L_C_R_bool
253 \cs_new_protected:Npn \@@_define_L_C_R:
254 {
255   \newcolumntype L l
256   \newcolumntype C c
257   \newcolumntype R r
258 }
```

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.

```
259 \int_new:N \g_@@_env_int
```

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

```
260 \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.

```
261 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
262 { \int_use:N \g_@@_env_int }
```

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

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

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

```
265 \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).

```
266 \dim_new:N \l_@@_columns_width_dim
```

The following counters will be used to count the numbers of rows and columns of the array.

```
267 \int_new:N \g_@@_row_total_int
268 \int_new:N \g_@@_col_total_int
```

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

```
269 \tl_new:N \l_@@_cell_type_tl
270 \tl_set:Nn \l_@@_cell_type_tl { c }
```


When there is a mono-column block (created by the command `\Block`), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the `\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`.

```
271 \dim_new:N \g_@@_blocks_wd_dim
```

Idem pour the mono-row blocks.

```
272 \dim_new:N \g_@@_blocks_ht_dim
273 \dim_new:N \g_@@_blocks_dp_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`.

```
274 \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.

```
275 \bool_new:N \l_@@_in_env_bool
```

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

```
276 \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.

```
277 \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.

```
278 \dim_new:N \l_@@_tabular_width_dim
```

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

```
279 \bool_new:N \l_@@_Matrix_bool
```

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

```
280 \bool_new:N \g_@@_rotate_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 }`).

```
281 \tl_new:N \g_@@_aux_tl
```

```
282 \cs_new_protected:Npn \@@_test_if_math_mode:
283 {
284   \if_mode_math: \else:
285     \@@_fatal:n { Outside-math-mode }
286   \fi:
287 }
```

The letter used for the vlins which will be drawn only in the sub-matrices. `vlism` stands for *vertical lines in sub-matrices*.

```
288 \tl_new:N \l_@@_letter_vlism_tl
```

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

```
289 \seq_new:N \g_@@_cols_vlism_seq
```

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

```
290 \colorlet { nicematrix-last-col } { . }
291 \colorlet { nicematrix-last-row } { . }
```

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

```
292 \str_new:N \g_@@_name_env_str
```

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

```
293 \tl_set:Nn \g_@@_com_or_env_str { environment }
```

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

```
294 \cs_new:Npn \@@_full_name_env:
295 {
296   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
297   { command \space \c_backslash_str \g_@@_name_env_str }
298   { environment \space \{ \g_@@_name_env_str \} }
299 }
```

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`).

```
300 \tl_new:N \g_nicematrix_code_after_tl
```

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

```
301 \tl_new:N \l_@@_code_tl
```

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

```
302 \tl_new:N \g_@@_internal_code_after_tl
```

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

```
303 \int_new:N \l_@@_old_iRow_int
304 \int_new:N \l_@@_old_jCol_int
```

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

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

```
305 \tl_new:N \l_@@_rules_color_tl
```

The number of letters `X` in the preamble of the array.

```
306 \int_new:N \g_@@_nb_of_X_int
```

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 `\l_@@_x_columns_dim` will be the width of the `X`-columns. That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
307 \bool_new:N \l_@@_X_columns_aux_bool
308 \dim_new:N \l_@@_X_columns_dim
```

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 `\Hdotsfor` is used in that column.

```
309 \bool_new:N \g_@@_after_col_zero_bool
```

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

```
310 \bool_new:N \g_@@_row_of_col_done_bool
```

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

```
311 \bool_new:N \g_@@_not_empty_cell_bool
```

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

- A `code-before` written in the `aux` file by a previous run. When the `aux` file is read, this `code-before` is stored in `\g_@@_code_before_i_tl` (where *i* is the number of the environment) and, at the beginning of the environment, it will be put in `\l_@@_code_before_tl`.
- The final user can explicitly add material in `\l_@@_code_before_tl` by using the key `code-before` or the keyword `\CodeBefore` (with the keyword `\Body`).

```
312 \tl_new:N \l_@@_code_before_tl
```

```
313 \bool_new:N \l_@@_code_before_bool
```

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).

```
314 \tl_new:N \g_@@_row_style_tl
```

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

```
315 \dim_new:N \l_@@_x_initial_dim
```

```
316 \dim_new:N \l_@@_y_initial_dim
```

```
317 \dim_new:N \l_@@_x_final_dim
```

```
318 \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 (if they don't exist yet: that's why we use `\dim_zero_new:N`).

```
319 \dim_zero_new:N \l_tmpc_dim
```

```
320 \dim_zero_new:N \l_tmpd_dim
```

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

```
321 \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”.

```
322 \dim_new:N \g_@@_width_last_col_dim
```

```
323 \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}-{jmin}-{imax}-{jmax}-{options}-{contents}`.

The variable is global because it will be modified in the cells of the array.

```
324 \seq_new:N \g_@@_blocks_seq
```

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

```
325 \seq_new:N \g_@@_pos_of_blocks_seq
```

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

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

```
326 \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.

```
327 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners` (or the key `hvlines-except-corners`), all the cells which are in an (empty) corner will be stored in the following sequence.

```
328 \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`).

```
329 \seq_new:N \g_@@_submatrix_names_seq
```

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) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
330 \seq_new:N \g_@@_multicolumn_cells_seq
```

```
331 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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

```
332 \int_new:N \l_@@_row_min_int
```

```
333 \int_new:N \l_@@_row_max_int
```

```
334 \int_new:N \l_@@_col_min_int
```

```
335 \int_new:N \l_@@_col_max_int
```

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

```
336 \seq_new:N \g_@@_submatrix_seq
```

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

```
337 \int_new:N \g_@@_static_num_of_col_int
```

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

```
338 \tl_new:N \l_@@_fill_tl
339 \tl_new:N \l_@@_draw_tl
340 \seq_new:N \l_@@_tikz_seq
341 \clist_new:N \l_@@_borders_clist
342 \dim_new:N \l_@@_rounded_corners_dim
```

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

The following token list correspond to the key `color` of the command `\Block`.

```
343 \tl_new:N \l_@@_color_tl
```

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

```
344 \dim_new:N \l_@@_line_width_dim
```

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

```
345 \tl_new:N \l_@@_hpos_of_block_tl
346 \tl_set:Nn \l_@@_hpos_of_block_tl { c }
347 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

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

```
348 \tl_new:N \l_@@_vpos_of_block_tl
349 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

```
350 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the key `hvlines` of the command `\Block`.

```
351 \bool_new:N \l_@@_hvlines_block_bool
```

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

```
352 \int_new:N \g_@@_block_box_int

353 \dim_new:N \l_@@_submatrix_extra_height_dim
354 \dim_new:N \l_@@_submatrix_left_xshift_dim
355 \dim_new:N \l_@@_submatrix_right_xshift_dim
356 \clist_new:N \l_@@_hlines_clist
357 \clist_new:N \l_@@_vlines_clist
358 \clist_new:N \l_@@_submatrix_hlines_clist
359 \clist_new:N \l_@@_submatrix_vlines_clist
```

Variables for the exterior rows and columns

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

- **First row**

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

```
360 \int_new:N \l_@@_first_row_int
361 \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.

```
362 \int_new:N \l_@@_first_col_int
363 \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).

```
364 \int_new:N \l_@@_last_row_int
365 \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”.⁵⁴

```
366 \bool_new:N \l_@@_last_row_without_value_bool
```

Idem for `\l_@@_last_col_without_value_bool`

```
367 \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.

```
368 \int_new:N \l_@@_last_col_int
369 \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.

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

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

⁵⁴We 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.

The command `\tabularnote`

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

```
371 \newcounter { tabularnote }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
379 \AtBeginDocument
380 {
381   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
382   {
383     \NewDocumentCommand \tabularnote { m }
384     { \@@_error:n { enumitem-not-loaded } }
385   }
386   {
```

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.

```
387     \newlist { tabularnotes } { enumerate } { 1 }
388     \setlist [ tabularnotes ]
389     {
390       topsep = Opt ,
391       noitemsep ,
392       leftmargin = * ,
```

```

393         align = left ,
394         labelsep = Opt ,
395         label =
396         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
397     }
398     \newlist { tabularnotes* } { enumerate* } { 1 }
399     \setlist [ tabularnotes* ]
400     {
401         afterlabel = \nobreak ,
402         itemjoin = \quad ,
403         label =
404         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
405     }

```

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

Unfortunately, if the package `caption` is loaded, the command `\caption` evaluates its argument twice and since it is not aware (of course) of `\tabularnote`, the command `\tabularnote` is, in fact, not usable in `\caption` when `caption` is loaded.⁵⁵

```

406     \NewDocumentCommand \tabularnote { m }
407     {
408         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
409         { \@@_error:n { tabularnote~forbidden } }
410         {

```

`\l_@@_number_of_notes_int` is used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. We will have to compose the labels of these notes as a comma separated list (e.g. a,b,c).

```

411         \int_incr:N \l_@@_number_of_notes_int

```

We expand the content of the note at the point of use of `\tabularnote` as does `\footnote`.

```

412         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
413         \peek_meaning:NF \tabularnote
414         {

```

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.

```

415         \hbox_set:Nn \l_tmpa_box
416         {

```

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

```

417         \@@_notes_label_in_tabular:n
418         {
419             \stepcounter { tabularnote }
420             \@@_notes_style:n { tabularnote }
421             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
422             {
423                 ,
424                 \stepcounter { tabularnote }
425                 \@@_notes_style:n { tabularnote }
426             }
427         }
428     }

```

⁵⁵We should try to find a solution to that problem.

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.

```

429         \addtocounter { tabularnote } { -1 }
430         \refstepcounter { tabularnote }
431         \int_zero:N \l_@@_number_of_notes_int
432         \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).

```

433         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
434     }
435 }
436 }
437 }
438 }

```

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.

```

439 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
440 {
441     \begin { pgfscope }
442     \pgfset
443     {
444         outer~sep = \c_zero_dim ,
445         inner~sep = \c_zero_dim ,
446         minimum~size = \c_zero_dim
447     }
448     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
449     \pgfnode
450     { rectangle }
451     { center }
452     {
453         \vbox_to_ht:nn
454         { \dim_abs:n { #5 - #3 } }
455         {
456             \vfill
457             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
458         }
459     }
460     { #1 }
461     { }
462     \end { pgfscope }
463 }

```

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.

```

464 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
465 {
466     \begin { pgfscope }
467     \pgfset
468     {
469         outer~sep = \c_zero_dim ,
470         inner~sep = \c_zero_dim ,
471         minimum~size = \c_zero_dim
472     }
473     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }

```

```

474 \pgfpointdiff { #3 } { #2 }
475 \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
476 \pgfnode
477 { rectangle }
478 { center }
479 {
480   \vbox_to_ht:nn
481   { \dim_abs:n \l_tmpb_dim }
482   { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
483 }
484 { #1 }
485 { }
486 \end { pgfscope }
487 }

```

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.

```

488 \bool_new:N \l_@@_colortbl_like_bool

```

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

```

489 \bool_new:N \l_@@_standard_cline_bool

```

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

```

490 \dim_new:N \l_@@_cell_space_top_limit_dim
491 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

492 \dim_new:N \l_@@_inter_dots_dim
493 \AtBeginDocument { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }

```

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

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

```

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

```

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

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

```

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

```

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

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

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

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

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

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

```
502 \tl_new:N \l_@@_baseline_tl
503 \tl_set:Nn \l_@@_baseline_tl c
```

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

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

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

```
505 \bool_new:N \l_@@_parallelize_diags_bool
506 \bool_set_true:N \l_@@_parallelize_diags_bool
```

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

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

```
508 \dim_new:N \l_@@_notes_above_space_dim
509 \AtBeginDocument { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
510 \bool_new:N \l_@@_nullify_dots_bool
```

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

```
511 \bool_new:N \l_@@_auto_columns_width_bool
```

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

```
512 \bool_new:N \g_@@_recreate_cell_nodes_bool
```

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

```
513 \str_new:N \l_@@_name_str
```

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

```
514 \bool_new:N \l_@@_medium_nodes_bool
515 \bool_new:N \l_@@_large_nodes_bool
```

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

```
516 \bool_new:N \l_@@_except_borders_bool
```

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

```
517 \dim_new:N \l_@@_left_margin_dim
518 \dim_new:N \l_@@_right_margin_dim
```

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

```
519 \dim_new:N \l_@@_extra_left_margin_dim
520 \dim_new:N \l_@@_extra_right_margin_dim
```

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

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

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

```
523 \tl_new:N \l_@@_xdots_color_tl
```

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

```
524 \tl_new:N \l_@@_delimiters_color_tl
```

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is fonction 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.

```
525 \bool_new:N \l_@@_delimiters_max_width_bool
```

```
526 \keys_define:nn { NiceMatrix / xdots }
527 {
528   line-style .code:n =
529   {
530     \bool_lazy_or:nnTF
```

We can’t use `\c_@@_tikz_loaded_bool` to test whether `tikz` is loaded because `\NiceMatrixOptions` may be used in the preamble of the document.

```
531   { \cs_if_exist_p:N \tikzpicture }
532   { \str_if_eq_p:nn { #1 } { standard } }
533   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
534   { \@@_error:n { bad-option~for~line-style } }
535   } ,
536   line-style .value_required:n = true ,
537   color .tl_set:N = \l_@@_xdots_color_tl ,
538   color .value_required:n = true ,
539   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
540   shorten .value_required:n = true ,
```

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

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

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

```
543   draw-first .code:n = \prg_do_nothing: ,
544   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
545 }
```

```
546 \keys_define:nn { NiceMatrix / rules }
547 {
548   color .tl_set:N = \l_@@_rules_color_tl ,
549   color .value_required:n = true ,
550   width .dim_set:N = \arrayrulewidth ,
551   width .value_required:n = true
552 }
```

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

```
553 \keys_define:nn { NiceMatrix / Global }
554 {
555   delimiters .code:n =
556     \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
557   delimiters .value_required:n = true ,
558   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
559   rules .value_required:n = true ,
560   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
561   standard-cline .default:n = true ,
562   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
563   cell-space-top-limit .value_required:n = true ,
564   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
565   cell-space-bottom-limit .value_required:n = true ,
566   cell-space-limits .meta:n =
567     {
568       cell-space-top-limit = #1 ,
569       cell-space-bottom-limit = #1 ,
570     } ,
571   cell-space-limits .value_required:n = true ,
572   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
573   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
574   light-syntax .default:n = true ,
575   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
576   end-of-row .value_required:n = true ,
577   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
578   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
579   last-row .int_set:N = \l_@@_last_row_int ,
580   last-row .default:n = -1 ,
581   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
582   code-for-first-col .value_required:n = true ,
583   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
584   code-for-last-col .value_required:n = true ,
585   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
586   code-for-first-row .value_required:n = true ,
587   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
588   code-for-last-row .value_required:n = true ,
589   hlines .clist_set:N = \l_@@_hlines_clist ,
590   vlines .clist_set:N = \l_@@_vlines_clist ,
591   hlines .default:n = all ,
592   vlines .default:n = all ,
```

```

593 vlines-in-sub-matrix .code:n =
594 {
595     \tl_if_single_token:nTF { #1 }
596     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
597     { \@@_error:n { One~letter~allowed } }
598 },
599 vlines-in-sub-matrix .value_required:n = true ,
600 hvlines .code:n =
601 {
602     \clist_set:Nn \l_@@_vlines_clist { all }
603     \clist_set:Nn \l_@@_hlines_clist { all }
604 },
605 hvlines-except-borders .code:n =
606 {
607     \clist_set:Nn \l_@@_vlines_clist { all }
608     \clist_set:Nn \l_@@_hlines_clist { all }
609     \bool_set_true:N \l_@@_except_borders_bool
610 },
611 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

612 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
613 renew-dots .value_forbidden:n = true ,
614 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
615 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
616 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
617 create-extra-nodes .meta:n =
618 { create-medium-nodes , create-large-nodes } ,
619 left-margin .dim_set:N = \l_@@_left_margin_dim ,
620 left-margin .default:n = \arraycolsep ,
621 right-margin .dim_set:N = \l_@@_right_margin_dim ,
622 right-margin .default:n = \arraycolsep ,
623 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
624 margin .default:n = \arraycolsep ,
625 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
626 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
627 extra-margin .meta:n =
628 { extra-left-margin = #1 , extra-right-margin = #1 } ,
629 extra-margin .value_required:n = true ,
630 }

```

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

```

631 \keys_define:nn { NiceMatrix / Env }
632 {
633

```

The key `hvlines-except-corners` is now deprecated.

```

634 hvlines-except-corners .code:n =
635 {
636     \clist_set:Nn \l_@@_corners_clist { #1 }
637     \clist_set:Nn \l_@@_vlines_clist { all }
638     \clist_set:Nn \l_@@_hlines_clist { all }
639 },
640 hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
641 corners .clist_set:N = \l_@@_corners_clist ,
642 corners .default:n = { NW , SW , NE , SE } ,
643 code-before .code:n =
644 {
645     \tl_if_empty:nF { #1 }
646     {

```

```

647         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
648         \bool_set_true:N \l_@@_code_before_bool
649     }
650 } ,

```

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

```

651 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
652 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
653 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
654 baseline .tl_set:N = \l_@@_baseline_tl ,
655 baseline .value_required:n = true ,
656 columns-width .code:n =
657     \tl_if_eq:nnTF { #1 } { auto }
658     { \bool_set_true:N \l_@@_auto_columns_width_bool }
659     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
660 columns-width .value_required:n = true ,
661 name .code:n =

```

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

```

662 \legacy_if:nF { measuring@ }
663 {
664     \str_set:Nn \l_tmpa_str { #1 }
665     \seq_if_in:NVTf \g_@@_names_seq \l_tmpa_str
666     { \@@_error:nn { Duplicate-name } { #1 } }
667     { \seq_gput_left:Nv \g_@@_names_seq \l_tmpa_str }
668     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
669 } ,
670 name .value_required:n = true ,
671 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
672 code-after .value_required:n = true ,
673 colortbl-like .code:n =
674     \bool_set_true:N \l_@@_colortbl_like_bool
675     \bool_set_true:N \l_@@_code_before_bool ,
676 colortbl-like .value_forbidden:n = true
677 }
678 \keys_define:nn { NiceMatrix / notes }
679 {
680     para .bool_set:N = \l_@@_notes_para_bool ,
681     para .default:n = true ,
682     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
683     code-before .value_required:n = true ,
684     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
685     code-after .value_required:n = true ,
686     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
687     bottomrule .default:n = true ,
688     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
689     style .value_required:n = true ,
690     label-in-tabular .code:n =
691         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
692     label-in-tabular .value_required:n = true ,
693     label-in-list .code:n =
694         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
695     label-in-list .value_required:n = true ,
696     enumitem-keys .code:n =
697     {
698         \bool_if:Ntf \c_@@_in_preamble_bool
699         {
700             \AtBeginDocument
701             {
702                 \bool_if:NT \c_@@_enumitem_loaded_bool
703                 { \setlist* [ tabularnotes ] { #1 } }

```

```

704     }
705   }
706   {
707     \bool_if:NT \c_@@_enumitem_loaded_bool
708     { \setlist* [ tabularnotes ] { #1 } }
709   }
710 },
711 enumitem-keys .value_required:n = true ,
712 enumitem-keys-para .code:n =
713 {
714   \bool_if:NTF \c_@@_in_preamble_bool
715   {
716     \AtBeginDocument
717     {
718       \bool_if:NT \c_@@_enumitem_loaded_bool
719       { \setlist* [ tabularnotes* ] { #1 } }
720     }
721   }
722   {
723     \bool_if:NT \c_@@_enumitem_loaded_bool
724     { \setlist* [ tabularnotes* ] { #1 } }
725   }
726 },
727 enumitem-keys-para .value_required:n = true ,
728 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
729 }
730 \keys_define:nn { NiceMatrix / delimiters }
731 {
732   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
733   max-width .default:n = true ,
734   color .tl_set:N = \l_@@_delimiters_color_tl ,
735   color .value_required:n = true ,
736 }

```

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

```

737 \keys_define:nn { NiceMatrix }
738 {
739   NiceMatrixOptions .inherit:n =
740   { NiceMatrix / Global } ,
741   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
742   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
743   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
744   NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
745   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
746   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
747   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
748   NiceMatrix .inherit:n =
749   {
750     NiceMatrix / Global ,
751     NiceMatrix / Env ,
752   } ,
753   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
754   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
755   NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
756   NiceTabular .inherit:n =
757   {
758     NiceMatrix / Global ,
759     NiceMatrix / Env
760   } ,
761   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
762   NiceTabular / rules .inherit:n = NiceMatrix / rules ,

```



```

763 NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
764 NiceArray .inherit:n =
765 {
766     NiceMatrix / Global ,
767     NiceMatrix / Env ,
768 } ,
769 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
770 NiceArray / rules .inherit:n = NiceMatrix / rules ,
771 NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
772 pNiceArray .inherit:n =
773 {
774     NiceMatrix / Global ,
775     NiceMatrix / Env ,
776 } ,
777 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
778 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
779 pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
780 }

```

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

```

781 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
782 {
783     last-col .code:n = \tl_if_empty:nF { #1 }
784                 { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
785                 \int_zero:N \l_@@_last_col_int ,
786     small .bool_set:N = \l_@@_small_bool ,
787     small .value_forbidden:n = true ,

```

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

```

788 renew-matrix .code:n = \@@_renew_matrix: ,
789 renew-matrix .value_forbidden:n = true ,

```

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

```

790 transparent .code:n =
791 {
792     \@@_renew_matrix:
793     \bool_set_true:N \l_@@_renew_dots_bool
794     \@@_error:n { Key~transparent }
795 } ,
796 transparent .value_forbidden:n = true,

```

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

```

797 exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

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

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

```

798 columns-width .code:n =
799     \tl_if_eq:nnTF { #1 } { auto }
800     { \@@_error:n { Option~auto~for~columns~width } }
801     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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.

```

802 allow-duplicate-names .code:n =
803     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
804 allow-duplicate-names .value_forbidden:n = true ,

```

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

```

805   letter-for-dotted-lines .code:n =
806   {
807     \tl_if_single_token:nTF { #1 }
808     { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
809     { \@@_error:n { One~letter~allowed } }
810   } ,
811   letter-for-dotted-lines .value_required:n = true ,
812   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
813   notes .value_required:n = true ,
814   sub-matrix .code:n =
815   \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
816   sub-matrix .value_required:n = true ,
817   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
818 }

819 \str_new:N \l_@@_letter_for_dotted_lines_str
820 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str

```

`\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.

```

821 \NewDocumentCommand \NiceMatrixOptions { m }
822 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

823 \keys_define:nn { NiceMatrix / NiceMatrix }
824 {
825   last-col .code:n = \tl_if_empty:nTF {#1}
826   {
827     \bool_set_true:N \l_@@_last_col_without_value_bool
828     \int_set:Nn \l_@@_last_col_int { -1 }
829   }
830   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
831   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
832   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
833   small .bool_set:N = \l_@@_small_bool ,
834   small .value_forbidden:n = true ,
835   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
836 }

```

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

```

837 \keys_define:nn { NiceMatrix / NiceArray }
838 {

```

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.

```

839   small .bool_set:N = \l_@@_small_bool ,
840   small .value_forbidden:n = true ,
841   last-col .code:n = \tl_if_empty:nF { #1 }
842   { \@@_error:n { last-col~non-empty~for~NiceArray } }
843   \int_zero:N \l_@@_last_col_int ,
844   notes / para .bool_set:N = \l_@@_notes_para_bool ,
845   notes / para .default:n = true ,
846   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
847   notes / bottomrule .default:n = true ,
848   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,

```

```

849   tabularnote .value_required:n = true ,
850   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
851   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
852   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
853 }
854 \keys_define:nn { NiceMatrix / pNiceArray }
855 {
856   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
857   last-col .code:n = \tl_if_empty:nF {#1}
858     { \@@_error:n { last-col~non~empty~for~NiceArray } }
859     \int_zero:N \l_@@_last_col_int ,
860   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
861   small .bool_set:N = \l_@@_small_bool ,
862   small .value_forbidden:n = true ,
863   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
864   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
865   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
866 }

```

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

```

867 \keys_define:nn { NiceMatrix / NiceTabular }
868 {
869   notes / para .bool_set:N = \l_@@_notes_para_bool ,
870   notes / para .default:n = true ,
871   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
872   notes / bottomrule .default:n = true ,
873   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
874   tabularnote .value_required:n = true ,
875   last-col .code:n = \tl_if_empty:nF {#1}
876     { \@@_error:n { last-col~non~empty~for~NiceArray } }
877     \int_zero:N \l_@@_last_col_int ,
878   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
879   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
880   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
881 }

```

Important code used by {NiceArrayWithDelims}

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

```

882 \cs_new_protected:Npn \@@_Cell:
883 {

```

The token list \g_@@_post_treatment_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-treatment*).

```

884   \tl_gclear:N \g_@@_post_treatment_cell_tl

```

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

```

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

```

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

```

886   \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.

```
887 \int_compare:nNnT \c@jCol = 1
888 { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_begin_of_row: }
```

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

```
889 \hbox_set:Nw \l_@@_cell_box
890 \bool_if:NF \l_@@_NiceTabular_bool
891 {
892   \c_math_toggle_token
893   \bool_if:NT \l_@@_small_bool \scriptstyle
894 }
```

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.

```
895 \g_@@_row_style_tl
896 \int_compare:nNnTF \c@iRow = 0
897 {
898   \int_compare:nNnT \c@jCol > 0
899   {
900     \l_@@_code_for_first_row_tl
901     \xglobal \colorlet { nicematrix-first-row } { . }
902   }
903 }
904 {
905   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
906   {
907     \l_@@_code_for_last_row_tl
908     \xglobal \colorlet { nicematrix-last-row } { . }
909   }
910 }
911 }
```

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.

```
912 \cs_new_protected:Npn \@@_begin_of_row:
913 {
914   \tl_gclear:N \g_@@_row_style_tl
915   \int_gincr:N \c@iRow
916   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
917   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
918   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
919   \pgfpicture
920   \pgfrememberpicturepositiononpagetrue
921   \pgfcoordinate
922   { \@@_env: - row - \int_use:N \c@iRow - base }
923   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
924   \str_if_empty:NF \l_@@_name_str
925   {
926     \pgfnodealias
927     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
928     { \@@_env: - row - \int_use:N \c@iRow - base }
929   }
930   \endpgfpicture
931 }
```

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.

```

932 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
933 {
934   \int_compare:nNnTF \c@iRow = 0
935   {
936     \dim_gset:Nn \g_@@_dp_row_zero_dim
937     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
938     \dim_gset:Nn \g_@@_ht_row_zero_dim
939     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
940   }
941   {
942     \int_compare:nNnT \c@iRow = 1
943     {
944       \dim_gset:Nn \g_@@_ht_row_one_dim
945       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
946     }
947   }
948 }

949 \cs_new_protected:Npn \@@_rotate_cell_box:
950 {
951   \box_rotate:Nn \l_@@_cell_box { 90 }
952   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
953   {
954     \vbox_set_top:Nn \l_@@_cell_box
955     {
956       \vbox_to_zero:n { }
957       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
958       \box_use:N \l_@@_cell_box
959     }
960   }
961   \bool_gset_false:N \g_@@_rotate_bool
962 }

963 \cs_new_protected:Npn \@@_adjust_size_box:
964 {
965   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
966   {
967     \box_set_wd:Nn \l_@@_cell_box
968     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
969     \dim_gzero:N \g_@@_blocks_wd_dim
970   }
971   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
972   {
973     \box_set_dp:Nn \l_@@_cell_box
974     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
975     \dim_gzero:N \g_@@_blocks_dp_dim
976   }
977   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
978   {
979     \box_set_ht:Nn \l_@@_cell_box
980     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
981     \dim_gzero:N \g_@@_blocks_ht_dim
982   }
983 }

984 \cs_new_protected:Npn \@@_end_Cell:
985 {
986   \@@_math_toggle_token:
987   \hbox_set_end:

```

The token list `\g_@@_post_treatment_cell_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

988 \g_@@_post_treatment_cell_tl
989 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
990 \@@_adjust_size_box:
991 \box_set_ht:Nn \l_@@_cell_box
992   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
993 \box_set_dp:Nn \l_@@_cell_box
994   { \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”).

```

995 \dim_gset:Nn \g_@@_max_cell_width_dim
996   { \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”.

```

997 \@@_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 technic:

- 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.

```

998 \bool_if:NTF \g_@@_empty_cell_bool
999   { \box_use_drop:N \l_@@_cell_box }
1000   {
1001     \bool_lazy_or:nnTF
1002       \g_@@_not_empty_cell_bool
1003       { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1004       \@@_node_for_cell:
1005       { \box_use_drop:N \l_@@_cell_box }
1006   }
1007 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1008 \bool_gset_false:N \g_@@_empty_cell_bool
1009 \bool_gset_false:N \g_@@_not_empty_cell_bool
1010 }

```

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

```

1011 \cs_new_protected:Npn \@@_node_for_cell:
1012   {
1013     \pgfpicture
1014     \pgfsetbaseline \c_zero_dim
1015     \pgfrememberpicturepositiononpagetrue
1016     \pgfset
1017     {
1018       inner-sep = \c_zero_dim ,
1019       minimum-width = \c_zero_dim
1020     }
1021     \pgfnode
1022     { rectangle }

```

```

1023     { base }
1024     { \box_use_drop:N \l_@@_cell_box }
1025     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1026     { }
1027     \str_if_empty:NF \l_@@_name_str
1028     {
1029         \pgfnodealias
1030         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1031         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1032     }
1033     \endpgfpicture
1034 }

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

1035 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1036 {
1037     \cs_new_protected:Npn \@@_patch_node_for_cell:
1038     {
1039         \hbox_set:Nn \l_@@_cell_box
1040         {
1041             \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1042             \hbox_overlap_left:n
1043             {
1044                 \pgfsys@markposition
1045                 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way latex, divps, ps2pdf (or Adobe Distiller). However, it seems to work.

```

1046             #1
1047         }
1048         \box_use:N \l_@@_cell_box
1049         \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1050         \hbox_overlap_left:n
1051         {
1052             \pgfsys@markposition
1053             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1054             #1
1055         }
1056     }
1057 }
1058 }

```

We have no explanation for the different behaviour between the TeX engines...

```

1059 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1060 {
1061     \@@_patch_node_for_cell:n
1062     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1063 }
1064 { \@@_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}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \end{pmatrix}$$

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.

```
1065 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3  
1066 {  
1067   \bool_if:NTF { #1 } \tl_gput_left:cx \tl_gput_right:cx  
1068     { g_@@_ #2 _ lines _ tl }  
1069     {  
1070       \use:c { @@ _ draw _ #2 : nnn }  
1071       { \int_use:N \c@iRow }  
1072       { \int_use:N \c@jCol }  
1073       { \exp_not:n { #3 } }  
1074     }  
1075 }
```

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

```
1076 \cs_new_protected:Npn \@@_revtex_array:  
1077 {  
1078   \cs_set_eq:NN \@acoll \@arrayacol  
1079   \cs_set_eq:NN \@acolr \@arrayacol  
1080   \cs_set_eq:NN \@acol \@arrayacol  
1081   \cs_set_nopar:Npn \@halignto { }  
1082   \@array@array  
1083 }  
  
1084 \cs_new_protected:Npn \@@_array:  
1085 {  
1086   \bool_if:NTF \c_@@_revtex_bool  
1087     \@@_revtex_array:  
1088     {  
1089       \bool_if:NTF \l_@@_NiceTabular_bool  
1090         { \dim_set_eq:NN \col@sep \tabcolsep }  
1091         { \dim_set_eq:NN \col@sep \arraycolsep }  
1092       \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim  
1093         { \cs_set_nopar:Npn \@halignto { } }  
1094         { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }  
1095     }  
1096 }
```

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

```
1095   \@tabarray  
1096 }
```

`\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.

```
1097   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]  
1098 }
```

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.

```
1099 \cs_set_eq:NN \@@_old_ialign: \ialign
```

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

```
1100 \cs_new_protected:Npn \@@_create_row_node:  
1101 {
```


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

```

1102   \hbox
1103   {
1104     \bool_if:NT \l_@@_code_before_bool
1105     {
1106       \vtop
1107       {
1108         \skip_vertical:N 0.5\arrayrulewidth
1109         \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1110         \skip_vertical:N -0.5\arrayrulewidth
1111       }
1112     }
1113     \pgfpicture
1114     \pgfrememberpicturerepositiononpagetrue
1115     \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1116     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1117     \str_if_empty:NF \l_@@_name_str
1118     {
1119       \pgfnodealias
1120       { \l_@@_name_str - row - \@@_succ:n \c@iRow }
1121       { \@@_env: - row - \@@_succ:n \c@iRow }
1122     }
1123     \endpgfpicture
1124   }
1125 }

```

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

```

1126 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1127 \cs_new_protected:Npn \@@_everycr_i:
1128 {
1129   \int_gzero:N \c@jCol
1130   \bool_gset_false:N \g_@@_after_col_zero_bool
1131   \bool_if:NF \g_@@_row_of_col_done_bool
1132   {
1133     \@@_create_row_node:

```

We don't draw the rules of the key hlines (or hvlines) but we reserve the vertical space for theses rules.

```

1134   \tl_if_empty:NF \l_@@_hlines_clist
1135   {
1136     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1137     {
1138       \exp_args:NNx
1139       \clist_if_in:NnT
1140       \l_@@_hlines_clist
1141       { \@@_succ:n \c@iRow }
1142     }
1143   }

```

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

```

1144   \int_compare:nNnT \c@iRow > { -1 }
1145   {
1146     \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

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

```

1147       { \hrule height \arrayrulewidth width \c_zero_dim }
1148     }
1149   }
1150 }
1151 }
1152 }

```

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

```

1153 \cs_set_protected:Npn \@@_newcolumntype #1
1154 {
1155   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1156   \peek_meaning:NTF [
1157     { \newcol@ #1 }
1158     { \newcol@ #1 [ 0 ] }
1159   }

```

When the key `renew-dots` is used, the following code will be executed.

```

1160 \cs_set_protected:Npn \@@_renew_dots:
1161 {
1162   \cs_set_eq:NN \ldots \@@_Ldots
1163   \cs_set_eq:NN \cdots \@@_Cdots
1164   \cs_set_eq:NN \vdots \@@_Vdots
1165   \cs_set_eq:NN \ddots \@@_Ddots
1166   \cs_set_eq:NN \iddots \@@_Iddots
1167   \cs_set_eq:NN \dots \@@_Ldots
1168   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1169 }

```

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

```

1170 \cs_new_protected:Npn \@@_colortbl_like:
1171 {
1172   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1173   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1174   \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1175 }

```

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

```

1176 \cs_new_protected:Npn \@@_pre_array_ii:
1177 {

```

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 ⁵⁶.

```

1178   \bool_if:NT \c_@@_booktabs_loaded_bool
1179     { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1180   \box_clear_new:N \l_@@_cell_box
1181   \normalbaselines

```

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}`).

```

1182   \bool_if:NT \l_@@_small_bool
1183     {
1184       \cs_set_nopar:Npn \arraystretch { 0.47 }
1185       \dim_set:Nn \arraycolsep { 1.45 pt }
1186     }

```

⁵⁶cf. `\nicematrix@redefine@check@rerun`

```

1187 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1188 {
1189     \tl_put_right:Nn \@@_begin_of_row:
1190     {
1191         \pgfsys@markposition
1192         { \@@_env: - row - \int_use:N \c@iRow - base }
1193     }
1194 }

```

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`.

```

1195 \cs_set_nopar:Npn \ialign
1196 {
1197     \bool_if:NTF \c_@@_colortbl_loaded_bool
1198     {
1199         \CT@everycr
1200         {
1201             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1202             \@@_everycr:
1203         }
1204     }
1205     { \everycr { \@@_everycr: } }
1206     \tabskip = \c_zero_skip

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁵⁷ 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`.

```

1207 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1208 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1209 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1210 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1211 \dim_gzero_new:N \g_@@_ht_row_one_dim
1212 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1213 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1214 \dim_gzero_new:N \g_@@_ht_last_row_dim
1215 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1216 \dim_gzero_new:N \g_@@_dp_last_row_dim
1217 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

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

```

1218 \cs_set_eq:NN \ialign \@@_old_ialign:
1219 \halign
1220 }

```

We keep in memory the old versions of `\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).

```

1221 \cs_set_eq:NN \@@_old_ldots \ldots
1222 \cs_set_eq:NN \@@_old_cdots \cdots
1223 \cs_set_eq:NN \@@_old_vdots \vdots
1224 \cs_set_eq:NN \@@_old_ddots \ddots
1225 \cs_set_eq:NN \@@_old_iddots \iddots
1226 \bool_if:NTF \l_@@_standard_cline_bool
1227 { \cs_set_eq:NN \cline \@@_standard_cline }

```

⁵⁷The option `small` of `nicematrix` changes (among other) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1228     { \cs_set_eq:NN \cline \@@_cline }
1229     \cs_set_eq:NN \ldots \@@_ldots
1230     \cs_set_eq:NN \Cdots \@@_Cdots
1231     \cs_set_eq:NN \Vdots \@@_Vdots
1232     \cs_set_eq:NN \Ddots \@@_Ddots
1233     \cs_set_eq:NN \Iddots \@@_Iddots
1234     \cs_set_eq:NN \hdottedline \@@_hdottedline:
1235     \cs_set_eq:NN \Hline \@@_Hline:
1236     \cs_set_eq:NN \Hspace \@@_Hspace:
1237     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1238     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1239     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1240     \cs_set_eq:NN \Block \@@_Block:
1241     \cs_set_eq:NN \rotate \@@_rotate:
1242     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1243     \cs_set_eq:NN \dotfill \@@_old_dotfill:
1244     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1245     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1246     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1247     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1248     \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1249     \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

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).

```

1250     \seq_gclear:N \g_@@_multicolumn_cells_seq
1251     \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

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).

```

1252     \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1253     \int_gzero_new:N \g_@@_row_total_int

```

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

```

1254     \int_gzero_new:N \g_@@_col_total_int
1255     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1256     \@@_renew_NC@rewrite@S:
1257     \bool_gset_false:N \g_@@_last_col_found_bool

```

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.

```

1258     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1259     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1260     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1261     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1262     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1263     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1264     \tl_gclear_new:N \g_nicematrix_code_before_tl
1265 }

```

This is the end of `\@@_pre_array_ii:`.

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

```

1266 \cs_new_protected:Npn \@@_pre_array:
1267 {

```

```

1268 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1269 \int_gzero_new:N \c@iRow
1270 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1271 \int_gzero_new:N \c@jCol

```

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).

```

1272 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1273 {
1274   \bool_set_true:N \l_@@_last_row_without_value_bool
1275   \bool_if:NT \g_@@_aux_found_bool
1276     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1277 }
1278 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1279 {
1280   \bool_if:NT \g_@@_aux_found_bool
1281     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1282 }

```

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

```

1283 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1284 {
1285   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1286     {
1287       \dim_gset:Nn \g_@@_ht_last_row_dim
1288         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1289       \dim_gset:Nn \g_@@_dp_last_row_dim
1290         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1291     }
1292 }

```

```

1293 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1294 \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.

```

1295 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the `aux` file.

```

1296 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1297 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The code in `\@@_pre_array_ii:` is used only here.

```

1298 \@@_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.

```

1299 \box_clear_new:N \l_@@_the_array_box

```

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

```
1300 \bool_if:NT \c_@@_define_L_C_R_bool \@@_define_L_C_R:
```

The preamble will be constructed in `\g_@@_preamble_tl`.

```
1301 \@@_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]`).

```
1302 \dim_zero_new:N \l_@@_left_delim_dim
1303 \dim_zero_new:N \l_@@_right_delim_dim
1304 \bool_if:NTF \l_@@_NiceArray_bool
1305 {
1306   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1307   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1308 }
1309 {
```

The command `\bBigg@` is a command of `amsmath`.

```
1310 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1311 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1312 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1313 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1314 }
```

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).

```
1315 \hbox_set:Nw \l_@@_the_array_box
1316 \skip_horizontal:N \l_@@_left_margin_dim
1317 \skip_horizontal:N \l_@@_extra_left_margin_dim
1318 \c_math_toggle_token
1319 \bool_if:NTF \l_@@_light_syntax_bool
1320 { \use:c { @@-light-syntax } }
1321 { \use:c { @@-normal-syntax } }
1322 }
```

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

```
1323 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1324 {
1325   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1326   \bool_set_true:N \l_@@_code_before_bool
```

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...

```
1327 \@@_pre_array:
1328 }
```

```
1329 \keys_define:nn { NiceMatrix / RowStyle }
1330 {
1331   cell-space-top-limit .code:n =
1332   {
1333     \tl_gput_right:Nn \g_@@_row_style_tl
1334     {
1335       \tl_gput_right:Nn \g_@@_post_treatment_cell_tl
1336       { \dim_set:Nn \l_@@_cell_space_top_limit_dim { #1 } }
1337     }
1338   }
```

```

1337     }
1338   } ,
1339   cell-space-top-limit .value_required:n = true ,
1340   cell-space-bottom-limit .code:n =
1341   {
1342     \tl_gput_right:Nn \g_@@_row_style_tl
1343     {
1344       \tl_gput_right:Nn \g_@@_post_treatment_cell_tl
1345       { \dim_set:Nn \l_@@_cell_space_bottom_limit_dim { #1 } }
1346     }
1347   } ,
1348   cell-space-bottom-limit .value_required:n = true ,
1349   cell-space-limits .meta:n =
1350   {
1351     cell-space-top-limit = #1 ,
1352     cell-space-bottom-limit = #1 ,
1353   } ,
1354   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
1355 }

1356 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
1357 {
1358   \tl_gset:Nn \g_@@_row_style_tl { #2 }
1359   \keys_set:nn { NiceMatrix / RowStyle } { #1 }
1360   #2
1361   \ignorespaces
1362 }

```

The \CodeBefore

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

```

1363 \cs_new_protected:Npn \@@_pre_code_before:
1364 {

```

First, we give values to the LaTeX counters iRow and jCol. We remind that, in the `code-before` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1365   \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1366   \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1367   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1368   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }

```

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

```

1369   \pgfsys@markposition { \@@_env: - position }
1370   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1371   \pgfpicture
1372   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1373   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1374   {
1375     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1376     \pgfcoordinate { \@@_env: - row - ##1 }
1377     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1378   }

```

Now, the recreation of the col nodes.

```

1379 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1380 {
1381   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1382   \pgfcoordinate { \@@_env: - col - ##1 }
1383   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1384 }

```

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

```

1385 \@@_create_diag_nodes:

```

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

```

1386 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1387 \endpgfpicture
1388 \bool_if:NT \c_@@_tikz_loaded_bool
1389 {
1390   \tikzset
1391   {
1392     every~picture / .style =
1393     { overlay , name~prefix = \@@_env: - }
1394   }
1395 }
1396 \cs_set_eq:NN \cellcolor \@@_cellcolor
1397 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1398 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1399 \cs_set_eq:NN \rowcolor \@@_rowcolor
1400 \cs_set_eq:NN \rowcolors \@@_rowcolors
1401 \cs_set_eq:NN \arraycolor \@@_arraycolor
1402 \cs_set_eq:NN \columncolor \@@_columncolor
1403 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1404 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1405 }

```

```

1406 \cs_new_protected:Npn \@@_exec_code_before:
1407 {
1408   \seq_gclear_new:N \g_@@_colors_seq
1409   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1410   \group_begin:

```

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

```

1411 \bool_if:NT \l_@@_NiceTabular_bool \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`.

```

1412 \exp_last_unbraced:N \@@_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.

```

1413 \@@_actually_color:
1414 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1415 \group_end:
1416 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1417 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1418 }

```



```

1419 \keys_define:nn { NiceMatrix / CodeBefore }
1420 {
1421   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1422   create-cell-nodes .default:n = true ,
1423   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1424   sub-matrix .value_required:n = true ,
1425   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1426   delimiters / color .value_required:n = true ,
1427   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1428 }
1429 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1430 {
1431   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1432   \@@_CodeBefore:w
1433 }

```

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.

```

1434 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1435 {
1436   \bool_if:NT \g_@@_aux_found_bool
1437   {
1438     \@@_pre_code_before:
1439     #1
1440   }
1441 }

```

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.

```

1442 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1443 {
1444   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1445   {
1446     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1447     \pgfcoordinate { \@@_env: - row - ##1 - base }
1448     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1449     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1450     {
1451       \cs_if_exist:cT
1452       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1453       {
1454         \pgfsys@getposition
1455         { \@@_env: - ##1 - #####1 - NW }
1456         \@@_node_position:
1457         \pgfsys@getposition
1458         { \@@_env: - ##1 - #####1 - SE }
1459         \@@_node_position_i:
1460         \@@_pgf_rect_node:nnn
1461         { \@@_env: - ##1 - #####1 }
1462         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1463         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1464       }
1465     }
1466   }
1467   \@@_create_extra_nodes:
1468 }

```

The environment {NiceArrayWithDelims}

```

1469 \NewDocumentEnvironment { NiceArrayWithDelims }
1470 { m m O { } m ! O { } t \CodeBefore }
1471 {
1472   \@@_provide_pgfsyspdfmark:
1473   \bool_if:NT \c_@@_footnote_bool \savenotes

```

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

```

1474   \bgroup

1475   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1476   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1477   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1478   \int_gzero:N \g_@@_block_box_int
1479   \dim_zero:N \g_@@_width_last_col_dim
1480   \dim_zero:N \g_@@_width_first_col_dim
1481   \bool_gset_false:N \g_@@_row_of_col_done_bool
1482   \str_if_empty:NT \g_@@_name_env_str
1483   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }

```

The following line will be deleted when we will consider that only versions of `siunitx` after v3.0 are compatible with `nicematrix`.

```

1484   \@@_adapt_S_column:
1485   \bool_if:NTF \l_@@_NiceTabular_bool
1486     \mode_leave_vertical:
1487     \@@_test_if_math_mode:
1488   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1489   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁵⁸. 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 beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

1490   \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

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

```

1491   \cs_if_exist:NT \tikz@library@external@loaded
1492   {
1493     \tikzexternaldisable
1494     \cs_if_exist:NT \ifstandalone
1495     { \tikzset { external / optimize = false } }
1496   }

```

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

```

1497   \int_gincr:N \g_@@_env_int
1498   \bool_if:NF \l_@@_block_auto_columns_width_bool
1499   { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1500   \seq_gclear:N \g_@@_blocks_seq
1501   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1502   \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq

```

⁵⁸e.g. `\color[rgb]{0.5,0.5,0}`

```

1503 \seq_gclear:N \g_@@_pos_of_xdots_seq
1504 \tl_gclear_new:N \g_@@_code_before_tl
1505 \tl_gclear:N \g_@@_row_style_tl

```

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

```

1506 \bool_gset_false:N \g_@@_aux_found_bool
1507 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1508 {
1509     \bool_gset_true:N \g_@@_aux_found_bool
1510     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1511 }

```

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.

```

1512 \tl_gclear:N \g_@@_aux_tl
1513 \tl_if_empty:NF \g_@@_code_before_tl
1514 {
1515     \bool_set_true:N \l_@@_code_before_bool
1516     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1517 }

```

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

```

1518 \bool_if:NTF \l_@@_NiceArray_bool
1519 { \keys_set:nn { NiceMatrix / NiceArray } }
1520 { \keys_set:nn { NiceMatrix / pNiceArray } }
1521 { #3 , #5 }

1522 \tl_if_empty:NF \l_@@_rules_color_tl
1523 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
1524 % \bigskip

```

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

```

1525 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:
1526 }
1527 {
1528     \bool_if:NTF \l_@@_light_syntax_bool
1529     { \use:c { end @@-light-syntax } }
1530     { \use:c { end @@-normal-syntax } }
1531     \c_math_toggle_token
1532     \skip_horizontal:N \l_@@_right_margin_dim
1533     \skip_horizontal:N \l_@@_extra_right_margin_dim
1534     \hbox_set_end:

```

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

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).

```

1535 \int_compare:nNnT \g_@@_nb_of_X_int > 0
1536 {
1537     \bool_if:NTF \l_@@_X_columns_aux_bool
1538     { \dim_set_eq:NN \l_tmpa_dim \l_@@_X_columns_dim }
1539     {
1540         \dim_set:Nn \l_tmpa_dim
1541         {
1542             ( \textwidth - \box_wd:N \l_@@_the_array_box )

```

```

1543         / \int_use:N \g_@@_nb_of_X_int
1544     }
1545 }
1546 \tl_gput_right:Nx \g_@@_aux_tl
1547 {
1548     \bool_set_true:N \l_@@_X_columns_aux_bool
1549     \dim_set:Nn \l_@@_X_columns_dim { \dim_use:N \l_tmpa_dim }
1550 }
1551 }

```

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).

```

1552 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1553 {
1554     \bool_if:NF \l_@@_last_row_without_value_bool
1555     {
1556         \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1557         {
1558             \@@_error:n { Wrong~last~row }
1559             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1560         }
1561     }
1562 }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.⁵⁹

```

1563 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1564 \bool_if:nTF \g_@@_last_col_found_bool
1565 { \int_gdecr:N \c@jCol }
1566 {
1567     \int_compare:nNnT \l_@@_last_col_int > { -1 }
1568     { \@@_error:n { last~col~not~used } }
1569 }

```

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

```

1570 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1571 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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. 116).

```

1572 \int_compare:nNnT \l_@@_first_col_int = 0
1573 {
1574     \skip_horizontal:N \col@sep
1575     \skip_horizontal:N \g_@@_width_first_col_dim
1576 }

```

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.

```

1577 \bool_if:NTF \l_@@_NiceArray_bool
1578 {
1579     \str_case:VnF \l_@@_baseline_tl
1580     {
1581         b \@@_use_arraybox_with_notes_b:
1582         c \@@_use_arraybox_with_notes_c:
1583     }
1584     \@@_use_arraybox_with_notes:
1585 }

```

⁵⁹We remind that the potential “first column” (exterior) has the number 0.

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).

```

1586 {
1587   \int_compare:nNnTF \l_@@_first_row_int = 0
1588   {
1589     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1590     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1591   }
1592   { \dim_zero:N \l_tmpa_dim }

```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.⁶⁰

```

1593   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1594   {
1595     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1596     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1597   }
1598   { \dim_zero:N \l_tmpb_dim }
1599   \hbox_set:Nn \l_tmpa_box
1600   {
1601     \c_math_toggle_token
1602     \tl_if_empty:NF \l_@@_delimiters_color_tl
1603     { \color { \l_@@_delimiters_color_tl } }
1604     \exp_after:wN \left \g_@@_left_delim_tl
1605     \vcenter
1606     {

```

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.

```

1607       \skip_vertical:N -\l_tmpa_dim
1608       \hbox
1609       {
1610         \bool_if:NTF \l_@@_NiceTabular_bool
1611         { \skip_horizontal:N -\tabcolsep }
1612         { \skip_horizontal:N -\arraycolsep }
1613         \@@_use_arraybox_with_notes_c:
1614         \bool_if:NTF \l_@@_NiceTabular_bool
1615         { \skip_horizontal:N -\tabcolsep }
1616         { \skip_horizontal:N -\arraycolsep }
1617       }

```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`).

```

1618       \skip_vertical:N -\l_tmpb_dim
1619     }

```

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.

```

1620     \tl_if_empty:NF \l_@@_delimiters_color_tl
1621     { \color { \l_@@_delimiters_color_tl } }
1622     \exp_after:wN \right \g_@@_right_delim_tl
1623     \c_math_toggle_token
1624   }

```

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.

```

1625   \bool_if:NTF \l_@@_delimiters_max_width_bool
1626   {
1627     \@@_put_box_in_flow_bis:nn
1628     \g_@@_left_delim_tl \g_@@_right_delim_tl
1629   }
1630   \@@_put_box_in_flow:

```

⁶⁰A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```
1631 }
```

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. 117).

```
1632 \bool_if:NT \g_@@_last_col_found_bool
1633 {
1634   \skip_horizontal:N \g_@@_width_last_col_dim
1635   \skip_horizontal:N \col@sep
1636 }
1637 \bool_if:NF \l_@@_Matrix_bool
1638 {
1639   \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1640   { \@@_error:n { columns-not-used } }
1641 }
1642 \group_begin:
1643 \globaldefs = 1
1644 \@@_msg_redirect_name:nn { columns-not-used } { error }
1645 \group_end:
1646 \@@_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.

```
1647 \egroup
```

We want to write on the aux file all the informations corresponding to the current environment.

```
1648 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1649 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1650 \iow_now:Nx \@mainaux
1651 {
1652   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1653   { \exp_not:V \g_@@_aux_tl }
1654 }
1655 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1656 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1657 }
```

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 `\g_@@_preamble_tl` and the modified version will be stored in `\g_@@_preamble_tl` also.

```
1658 \cs_new_protected:Npn \@@_construct_preamble:
1659 {
```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and with all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `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 `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```
1660 \group_begin:
```

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).

```

1661 \bool_if:NF \l_@@_Matrix_bool
1662 {
1663   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1664   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

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

```

1665 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

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

```

1666 \@tempswatrue

```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```

1667 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```

1668 \int_gzero:N \c@jCol
1669 \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1670 \bool_gset_false:N \g_tmpb_bool
1671 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1672 {
1673   \tl_gset:Nn \g_@@_preamble_tl
1674     { ! { \skip_horizontal:N \arrayrulewidth } }
1675 }
1676 {
1677   \clist_if_in:NnT \l_@@_vlines_clist 1
1678   {
1679     \tl_gset:Nn \g_@@_preamble_tl
1680       { ! { \skip_horizontal:N \arrayrulewidth } }
1681   }
1682 }

```

The number of letters `X` in the preamble of the array.

```

1683 \int_gzero:N \g_@@_nb_of_X_int

```

The sequence `\g_@@_cols_vlsim_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlsim`).

```

1684 \seq_clear:N \g_@@_cols_vlism_seq

```

The counter `\l_tmpa_int` will count the number of consecutive occurrences of the symbol `|`.

```

1685 \int_zero:N \l_tmpa_int

```

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

```

1686 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
1687 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1688 }

```

Now, we replace `\columncolor` by `\@@_columncolor_preamble`.

```

1689 \bool_if:NT \l_@@_colortbl_like_bool
1690 {
1691   \regex_replace_all:NnN
1692     \c_@@_columncolor_regex
1693     { \c { @@_columncolor_preamble } }
1694     \g_@@_preamble_tl
1695 }

```

Now, we can close the TeX group which was opened for the redefinition of the columns of type `w` and `W`.

```
1696 \group_end:
```

If there was delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```
1697 \bool_lazy_or:nnT
1698 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
1699 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
1700 { \bool_set_false:N \l_@@_NiceArray_bool }
```

We want to remind whether there is a specifier `|` at the end of the preamble.

```
1701 \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
```

We complete the preamble with the potential “exterior columns” (on both sides).

```
1702 \int_compare:nNnTF \l_@@_first_col_int = 0
1703 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1704 {
1705   \bool_lazy_all:nT
1706   {
1707     \l_@@_NiceArray_bool
1708     { \bool_not_p:n \l_@@_NiceTabular_bool }
1709     { \tl_if_empty_p:N \l_@@_vlines_clist }
1710     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1711   }
1712   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1713 }
1714 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1715 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1716 {
1717   \bool_lazy_all:nT
1718   {
1719     \l_@@_NiceArray_bool
1720     { \bool_not_p:n \l_@@_NiceTabular_bool }
1721     { \tl_if_empty_p:N \l_@@_vlines_clist }
1722     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1723   }
1724   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1725 }
```

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 `{NiceTabular*}` (`\l_@@_tabular_width_dim=0pt`).

```
1726 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1727 {
1728   \tl_gput_right:Nn \g_@@_preamble_tl
1729   { > { \@@_error_too_much_cols: } 1 }
1730 }
1731 }
```

```
1732 \cs_new_protected:Npn \@@_patch_preamble:n #1
1733 {
1734   \str_case:nnF { #1 }
1735   {
1736     c { \@@_patch_preamble_i:n #1 }
1737     l { \@@_patch_preamble_i:n #1 }
1738     r { \@@_patch_preamble_i:n #1 }
1739     > { \@@_patch_preamble_ii:nn #1 }
1740     ! { \@@_patch_preamble_ii:nn #1 }
1741     @ { \@@_patch_preamble_ii:nn #1 }
1742     | { \@@_patch_preamble_iii:n #1 }
1743     p { \@@_patch_preamble_iv:nnn t #1 }
```



```

1744 b { \@@_patch_preamble_iv:nnn b #1 }
1745 m { \@@_patch_preamble_iv:nnn b #1 }
1746 \@@_w: { \@@_patch_preamble_v:nnnn { } #1 }
1747 \@@_W: { \@@_patch_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1748 \@@_true_c: { \@@_patch_preamble_vi:n #1 }
1749 ( { \@@_patch_preamble_vii:nn #1 }
1750 [ { \@@_patch_preamble_vii:nn #1 }
1751 \{ { \@@_patch_preamble_vii:nn #1 }
1752 ) { \@@_patch_preamble_viii:nn #1 }
1753 ] { \@@_patch_preamble_viii:nn #1 }
1754 \} { \@@_patch_preamble_viii:nn #1 }
1755 X { \@@_patch_preamble_xii: }
1756 C { \@@_error:nn { old~column~type } #1 }
1757 L { \@@_error:nn { old~column~type } #1 }
1758 R { \@@_error:nn { old~column~type } #1 }
1759 \q_stop { }
1760 }
1761 {
1762 \str_if_eq:VnTF \l_@@_letter_for_dotted_lines_str { #1 }
1763 { \@@_patch_preamble_xi:n #1 }
1764 {
1765 \str_if_eq:VnTF \l_@@_letter_vlism_tl { #1 }
1766 {
1767 \seq_gput_right:Nx \g_@@_cols_vlism_seq
1768 { \int_eval:n { \c@jCol + 1 } }
1769 \tl_gput_right:Nx \g_@@_preamble_tl
1770 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1771 \@@_patch_preamble:n
1772 }
1773 {
1774 \bool_lazy_and:nnTF
1775 { \str_if_eq_p:nn { : } { #1 } }
1776 \c_@@_arydshln_loaded_bool
1777 {
1778 \tl_gput_right:Nn \g_@@_preamble_tl { : }
1779 \@@_patch_preamble:n
1780 }
1781 { \@@_fatal:nn { unknown~column~type } { #1 } }
1782 }
1783 }
1784 }
1785 }

```

For c, l and r

```

1786 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1787 {
1788 \tl_gput_right:Nn \g_@@_preamble_tl
1789 {
1790 > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
1791 #1
1792 < \@@_end_Cell:
1793 }

```

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

```

1794 \int_gincr:N \c@jCol
1795 \@@_patch_preamble_x:n
1796 }

```

For >, ! and @

```

1797 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1798 {
1799 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
1800 \@@_patch_preamble:n
1801 }

```

For l

```

1802 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1803 {
\l_tmpa_int is the number of successive occurrences of l
1804 \int_incr:N \l_tmpa_int
1805 \@@_patch_preamble_iii_i:n
1806 }

1807 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1808 {
1809 \str_if_eq:nnTF { #1 } l
1810 { \@@_patch_preamble_iii:n l }
1811 {
1812 \tl_gput_right:Nx \g_@@_preamble_tl
1813 {
1814 \exp_not:N !
1815 {
1816 \skip_horizontal:n
1817 {
1818 \dim_eval:n
1819 {
1820 \arrayrulewidth * \l_tmpa_int
1821 + \doublerulesep * ( \l_tmpa_int - 1)
1822 }
1823 }
1824 }
1825 }
1826 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1827 { \@@_vline:nn { \@@_succ:n \c@jCol } { \int_use:N \l_tmpa_int } }
1828 \int_zero:N \l_tmpa_int
1829 \str_if_eq:nnT { #1 } { \q_stop }
1830 { \bool_gset_true:N \g_tmpb_bool }
1831 \@@_patch_preamble:n #1
1832 }
1833 }

1834 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

For p and b

```

1835 \cs_new_protected:Npn \@@_patch_preamble_iv:nnn #1 #2 #3
1836 {
1837 \tl_gput_right:Nn \g_@@_preamble_tl
1838 {
1839 > {
1840 \@@_Cell:
1841 \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }

```

The following lines have been taken from array.sty.

```

1842 \everypar
1843 {
1844 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1845 \everypar { }
1846 }
1847 \g_@@_row_style_tl % added the 2021-07-17
1848 \arraybackslash
1849 }
1850 c
1851 < {

```

The following line has been taken from array.sty.

```

1852 \@finalstrut \@arstrutbox
1853 \end { minipage }

```

If the letter in the preamble is m, you require a post-treatment for the box \l_@@_cell_box.

```

1854         \str_if_eq:nnT { #2 } m \@@_center_cell_box:
1855         \@@_end_Cell:
1856     }
1857 }

```

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

```

1858     \int_gincr:N \c@jCol
1859     \@@_patch_preamble_x:n
1860 }

```

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, you consider (as in array) that if the height of the cell is no more that the height of \@arstrutbox, there is only one row.

```

1861 \cs_new_protected:Npn \@@_center_cell_box:
1862 {

```

By putting instructions in \g_@@_post_treatment_cell_tl, we require a post-treatment of the box \l_@@_cell_box.

```

1863     \tl_gput_right:Nn \g_@@_post_treatment_cell_tl
1864     {
1865         \int_compare:nNnT
1866         { \box_ht:N \l_@@_cell_box }
1867         >
1868         { \box_ht:N \@arstrutbox }
1869         {
1870             \hbox_set:Nn \l_@@_cell_box
1871             {
1872                 \box_move_down:nn
1873                 {
1874                     ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
1875                     + \baselineskip ) / 2
1876                 }
1877                 { \box_use:N \l_@@_cell_box }
1878             }
1879         }
1880     }
1881 }

```

For w and W

```

1882 \cs_new_protected:Npn \@@_patch_preamble_v:nnnn #1 #2 #3 #4
1883 {
1884     \tl_gput_right:Nn \g_@@_preamble_tl
1885     {
1886         > {
1887             \hbox_set:Nw \l_@@_cell_box
1888             \@@_Cell:
1889             \tl_set:Nn \l_@@_cell_type_tl { #3 }
1890         }
1891         c
1892         < {
1893             \@@_end_Cell:
1894             #1
1895             \hbox_set_end:
1896             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1897             \@@_adjust_size_box:
1898             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
1899         }
1900     }

```

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

```

1901 \int_gincr:N \c@jCol
1902 \@@_patch_preamble_x:n
1903 }

```

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

```

1904 \cs_new_protected:Npn \@@_patch_preamble_vi:n #1
1905 {
1906 \tl_gput_right:Nn \g_@@_preamble_tl { c }

```

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

```

1907 \int_gincr:N \c@jCol
1908 \@@_patch_preamble_x:n
1909 }

```

For (, [and \{.

```

1910 \cs_new_protected:Npn \@@_patch_preamble_vii:nn #1 #2
1911 {
1912 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in {NiceArray}, we reserve space for the left delimiter.

```

1913 \int_compare:nNnTF \c@jCol = \c_zero_int
1914 {
1915 \str_if_eq:VnTF \g_@@_left_delim_tl { . }
1916 {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

1917 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1918 \tl_gset:Nn \g_@@_right_delim_tl { . }
1919 \@@_patch_preamble:n #2
1920 }
1921 {
1922 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1923 \@@_patch_preamble_vii_i:nn { #1 } { #2 }
1924 }
1925 }
1926 { \@@_patch_preamble_vii_i:nn { #1 } { #2 } }
1927 }
1928 \cs_new_protected:Npn \@@_patch_preamble_vii_i:nn #1 #2
1929 {
1930 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1931 { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
1932 \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
1933 {
1934 \@@_error:nn { delimiter~after~opening } { #2 }
1935 \@@_patch_preamble:n
1936 }
1937 { \@@_patch_preamble:n #2 }
1938 }

```

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}).

```

1939 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
1940 {
1941 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
1942 \tl_if_in:nnTF { ) ] \} } { #2 }
1943 { \@@_patch_preamble_viii_i:nnn #1 #2 }
1944 {
1945 \tl_if_eq:nnTF { \q_stop } { #2 }
1946 {

```

```

1947 \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1948 { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
1949 {
1950 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1951 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1952 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1953 \@@_patch_preamble:n #2
1954 }
1955 }
1956 {
1957 \tl_if_in:nnT { ( [ \{ } { #2 }
1958 { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
1959 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1960 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1961 \@@_patch_preamble:n #2
1962 }
1963 }
1964 }
1965 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nnn #1 #2 #3
1966 {
1967 \tl_if_eq:nnTF { \q_stop } { #3 }
1968 {
1969 \str_if_eq:VnTF \g_@@_right_delim_tl { . }
1970 {
1971 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1972 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1973 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1974 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1975 }
1976 {
1977 \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
1978 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1979 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1980 \@@_error:nn { double-closing-delimiter } { #2 }
1981 }
1982 }
1983 {
1984 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1985 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
1986 \@@_error:nn { double-closing-delimiter } { #2 }
1987 \@@_patch_preamble:n #3
1988 }
1989 }
1990 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
1991 {
1992 \tl_gput_right:Nn \g_@@_preamble_tl
1993 { ! { \skip_horizontal:N 2\l_@@_radius_dim } }

```

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

```

1994 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1995 { \@@_vdottedline:n { \int_use:N \c@jCol } }
1996 \@@_patch_preamble:n
1997 }

```

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.

```

1998 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
1999 {
2000 \str_if_eq:nnTF { #1 } { < }
2001 \@@_patch_preamble_ix:n
2002 {
2003 \tl_if_eq:NnTF \l_@@_vlines_clist { all }

```

```

2004     {
2005         \tl_gput_right:Nn \g_@@_preamble_tl
2006         { ! { \skip_horizontal:N \arrayrulewidth } }
2007     }
2008     {
2009         \exp_args:NNx
2010         \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2011         {
2012             \tl_gput_right:Nn \g_@@_preamble_tl
2013             { ! { \skip_horizontal:N \arrayrulewidth } }
2014         }
2015     }
2016     \@@_patch_preamble:n { #1 }
2017 }
2018 }
2019 \cs_new_protected:Npn \@@_patch_preamble_ix:n #1
2020 {
2021     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2022     \@@_patch_preamble_x:n
2023 }

```

For the case of a letter X

```

2024 \cs_new_protected:Npn \@@_patch_preamble_xii:
2025 {
2026     \int_gincr:N \g_@@_nb_of_X_int

```

We test whether you know the width of the X-columns by reading the value in the aux file (of course, this is not possible at the first compilation).

```

2027     \bool_if:NTF \l_@@_X_columns_aux_bool
2028     { \exp_args:NNNV \@@_patch_preamble_iv:nnn t p \l_@@_X_columns_dim }
2029     {
2030         \tl_gput_right:Nn \g_@@_preamble_tl
2031         {
2032             > {
2033                 \@@_Cell:
2034                 \tl_set:Nn \l_@@_cell_type_tl { c }

```

The following code will nullify the box of the cell.

```

2035                 \tl_gput_right:Nn \g_@@_post_treatment_cell_tl
2036                 { \hbox_set:Nn \l_@@_cell_box { } }

```

The content of the cell is composed in \l_tmpa_box which will be discarded.

```

2037             }
2038             c
2039             < { \@@_end_Cell: }
2040         }
2041         \int_gincr:N \c@jCol
2042         \@@_patch_preamble_x:n
2043     }
2044 }

```

The redefinition of \multicolumn

The following command must *not* be protected since it begins with \multispan (a TeX primitive).

```

2045 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2046 {

```

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 than one column specifier in the preamble of `\multicolumn`.

```

2047 \multispan { #1 }
2048 \begingroup
2049 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

You do the expansion of the (small) preamble with the tools of `array`.

```

2050 \@temptokena = { #2 }
2051 \@tempswatrue
2052 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we patch the (small) preamble as we have done with the main preamble of the `array`.

```

2053 \tl_gclear:N \g_@@_preamble_tl
2054 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

The following lines are an adaptation of the definition of `\multicolumn` in `array`.

```

2055 \exp_args:NV \mkpream \g_@@_preamble_tl
2056 \@addtopreamble \empty
2057 \endgroup

```

Now, you do a treatment specific to `nicematrix` which has no equivalent in the original definition of `\multicolumn`.

```

2058 \int_compare:nNnT { #1 } > 1
2059 {
2060   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2061   { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
2062   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2063   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2064   {
2065     { \int_use:N \c@iRow }
2066     { \int_eval:n { \c@jCol + 1 } }
2067     { \int_use:N \c@iRow }
2068     { \int_eval:n { \c@jCol + #1 } }
2069   }
2070 }

```

The following lines were in the original definition of `\multicolumn`.

```

2071 \cs_set:Npn \@sharp { #3 }
2072 \@arstrut
2073 \@preamble
2074 \null

```

We add some lines.

```

2075 \int_gadd:Nn \c@jCol { #1 - 1 }
2076 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2077 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2078 \ignorespaces
2079 }

```

The following commands will patch the (small) preamble of the `\multicolumn`. All those commands have a `m` in their name to recall that they deal with the redefinition of `\multicolumn`.

```

2080 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2081 {
2082   \str_case:nnF { #1 }
2083   {
2084     c { \@@_patch_m_preamble_i:n #1 }
2085     l { \@@_patch_m_preamble_i:n #1 }
2086     r { \@@_patch_m_preamble_i:n #1 }
2087     > { \@@_patch_m_preamble_ii:nn #1 }
2088     ! { \@@_patch_m_preamble_ii:nn #1 }
2089     @ { \@@_patch_m_preamble_ii:nn #1 }

```

```

2090 | { \@@_patch_m_preamble_iii:n #1 }
2091 p { \@@_patch_m_preamble_iv:nnn t #1 }
2092 m { \@@_patch_m_preamble_iv:nnn c #1 }
2093 b { \@@_patch_m_preamble_iv:nnn b #1 }
2094 \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2095 \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2096 \@@_true_c: { \@@_patch_m_preamble_vi:n #1 }
2097 C { \@@_error:nn { old~column~type } #1 }
2098 L { \@@_error:nn { old~column~type } #1 }
2099 R { \@@_error:nn { old~column~type } #1 }
2100 \q_stop { }
2101 }
2102 { \@@_fatal:nn { unknown~column~type } { #1 } }
2103 }

```

For c, l and r

```

2104 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2105 {
2106   \tl_gput_right:Nn \g_@@_preamble_tl
2107   {
2108     > { \@@_Cell: \tl_set:Nn \l_@@_cell_type_tl { #1 } }
2109     #1
2110     < \@@_end_Cell:
2111   }

```

We test for the presence of a <.

```

2112 \@@_patch_m_preamble_x:n
2113 }

```

For >, ! and @

```

2114 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2115 {
2116   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2117   \@@_patch_m_preamble:n
2118 }

```

For |

```

2119 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2120 {
2121   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2122   \@@_patch_m_preamble:n
2123 }

```

For p, m and b

```

2124 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2125 {
2126   \tl_gput_right:Nn \g_@@_preamble_tl
2127   {
2128     > {
2129       \@@_Cell:
2130       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2131       \mode_leave_vertical:
2132       \arraybackslash
2133       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2134     }
2135     c
2136     < {
2137       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2138       \end { minipage }
2139       \@@_end_Cell:
2140     }
2141   }

```


We test for the presence of a <.

```
2142 \@@_patch_m_preamble_x:n
2143 }
```

For w and W

```
2144 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2145 {
2146   \tl_gput_right:Nn \g_@@_preamble_tl
2147   {
2148     > {
2149       \hbox_set:Nw \l_@@_cell_box
2150       \@@_Cell:
2151       \tl_set:Nn \l_@@_cell_type_tl { #3 }
2152     }
2153     c
2154     < {
2155       \@@_end_Cell:
2156       #1
2157       \hbox_set_end:
2158       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2159       \@@_adjust_size_box:
2160       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2161     }
2162   }
}
```

We test for the presence of a <.

```
2163 \@@_patch_m_preamble_x:n
2164 }
```

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

```
2165 \cs_new_protected:Npn \@@_patch_m_preamble_vi:n #1
2166 {
2167   \tl_gput_right:Nn \g_@@_preamble_tl { c }
```

We test for the presence of a <.

```
2168 \@@_patch_m_preamble_x:n
2169 }
```

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.

```
2170 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2171 {
2172   \str_if_eq:nnTF { #1 } { < }
2173   \@@_patch_m_preamble_ix:n
2174   {
2175     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2176     {
2177       \tl_gput_right:Nn \g_@@_preamble_tl
2178       { ! { \skip_horizontal:N \arrayrulewidth } }
2179     }
2180     {
2181       \exp_args:NNx
2182       \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2183       {
2184         \tl_gput_right:Nn \g_@@_preamble_tl
2185         { ! { \skip_horizontal:N \arrayrulewidth } }
2186       }
2187     }
2188     \@@_patch_m_preamble:n { #1 }
2189   }
2190 }
```

```

2191 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2192 {
2193   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2194   \@@_patch_m_preamble_x:n
2195 }

```

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

```

2196 \cs_new_protected:Npn \@@_put_box_in_flow:
2197 {
2198   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2199   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2200   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2201     { \box_use_drop:N \l_tmpa_box }
2202     \@@_put_box_in_flow_i:
2203 }

```

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).

```

2204 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2205 {
2206   \pgfpicture
2207     \@@_qpoint:n { row - 1 }
2208     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2209     \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2210     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2211     \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }

```

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

```

2212   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2213   {
2214     \int_set:Nn \l_tmpa_int
2215     {
2216       \str_range:Nnn
2217         \l_@@_baseline_tl
2218         6
2219         { \tl_count:V \l_@@_baseline_tl }
2220     }
2221     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2222   }
2223   {
2224     \str_case:VnF \l_@@_baseline_tl
2225     {
2226       { t } { \int_set:Nn \l_tmpa_int 1 }
2227       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2228     }
2229     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2230     \bool_lazy_or:nnT
2231     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2232     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2233     {
2234       \@@_error:n { bad~value~for~baseline }
2235       \int_set:Nn \l_tmpa_int 1
2236     }
2237     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2238     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2239   }
2240   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

Now, `\g_tmpa_dim` contains the value of the y translation we have to to.

```

2241   \endpgfpicture
2242   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2243   \box_use_drop:N \l_tmpa_box
2244 }

```

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).

```

2245 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2246 {

```

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.

```

2247   \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2248   {
2249     \box_set_wd:Nn \l_@@_the_array_box
2250     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2251   }

```

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

```

2252   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }

```

The `\hbox` avoids that the `pgfpicture` inside `\@@_draw_blocks` adds a extra vertical space before the notes.

```

2253   \hbox:n
2254   {
2255     \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.

```

2256     \@@_create_extra_nodes:
2257     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2258   }
2259   \bool_lazy_or:nnT
2260   { \int_compare_p:nNn \c@tabularnote > 0 }
2261   { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2262   \@@_insert_tabularnotes:
2263   \end { minipage }
2264 }
2265 \cs_new_protected:Npn \@@_insert_tabularnotes:
2266 {
2267   \skip_vertical:N 0.65ex

```

The TeX group is for potential specifications in the `\l_@@_notes_code_before_tl`.

```

2268   \group_begin:
2269   \l_@@_notes_code_before_tl
2270   \tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }

```

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

```

2271   \int_compare:nNnT \c@tabularnote > 0
2272   {
2273     \bool_if:NTF \l_@@_notes_para_bool
2274     {
2275       \begin { tabularnotes* }
2276       \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2277       \end { tabularnotes* }

```

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

```

2278         \par
2279     }
2280     {
2281         \tabularnotes
2282         \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2283         \endtabularnotes
2284     }
2285 }
2286 \unskip
2287 \group_end:
2288 \bool_if:NT \l_@@_notes_bottomrule_bool
2289 {
2290     \bool_if:NTF \c_@@_booktabs_loaded_bool
2291     {

```

The two dimensions `\aboverulesep` et `\heavyrulewidth` are parameters defined by `booktabs`.

```

2292         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2293     { \CT@arc@ \hrule height \heavyrulewidth }
2294 }
2295 { \@@_error:n { bottomrule~without~booktabs } }
2296 }
2297 \l_@@_notes_code_after_tl
2298 \seq_gclear:N \g_@@_tabularnotes_seq
2299 \int_gzero:N \c@tabularnote
2300 }

```

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`.

```

2301 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2302 {
2303     \pgfpicture
2304     \@@_qpoint:n { row - 1 }
2305     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2306     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2307     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2308     \endpgfpicture
2309     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2310     \int_compare:nNnT \l_@@_first_row_int = 0
2311     {
2312         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2313         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2314     }
2315     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2316 }

```

Now, the general case.

```

2317 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2318 {

```

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

```

2319     \tl_if_eq:NnT \l_@@_baseline_tl { t }
2320     { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

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

```

2321     \pgfpicture
2322     \@@_qpoint:n { row - 1 }
2323     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2324     \str_if_in:NnTF \l_@@_baseline_tl { line- }

```

```

2325 {
2326   \int_set:Nn \l_tmpa_int
2327   {
2328     \str_range:Nnn
2329       \l_@@_baseline_tl
2330       6
2331       { \tl_count:V \l_@@_baseline_tl }
2332   }
2333   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2334 }
2335 {
2336   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2337   \bool_lazy_or:nnT
2338     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2339     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2340   {
2341     \@@_error:n { bad~value~for~baseline }
2342     \int_set:Nn \l_tmpa_int 1
2343   }
2344   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2345 }
2346 \dim_gsub:Nn \g_tmpa_dim \pgf@y
2347 \endpgfpicture
2348 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2349 \int_compare:nNnT \l_@@_first_row_int = 0
2350 {
2351   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2352   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2353 }
2354 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2355 }

```

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.

```

2356 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2357 {

```

We will compute the real width of both delimiters used.

```

2358   \dim_zero_new:N \l_@@_real_left_delim_dim
2359   \dim_zero_new:N \l_@@_real_right_delim_dim
2360   \hbox_set:Nn \l_tmpb_box
2361   {
2362     \c_math_toggle_token
2363     \left #1
2364     \vcenter
2365     {
2366       \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2367       { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2368       { }
2369     }
2370     \right .
2371     \c_math_toggle_token
2372   }
2373   \dim_set:Nn \l_@@_real_left_delim_dim
2374   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2375   \hbox_set:Nn \l_tmpb_box
2376   {
2377     \c_math_toggle_token
2378     \left .
2379     \vbox_to_ht:nn

```

Here, you should use `\box_ht_plus_dp:N` when TeXLive 2021 will be available on Overleaf.

```

2380     { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
2381     { }
2382     \right #2
2383     \c_math_toggle_token
2384   }
2385   \dim_set:Nn \l_@@_real_right_delim_dim
2386     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2387   \skip_horizontal:N \l_@@_left_delim_dim
2388   \skip_horizontal:N -\l_@@_real_left_delim_dim
2389   \@@_put_box_in_flow:
2390   \skip_horizontal:N \l_@@_right_delim_dim
2391   \skip_horizontal:N -\l_@@_real_right_delim_dim
2392 }

```

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

```

2393 \NewDocumentEnvironment { @@-normal-syntax } { }

```

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

```

2394 {
2395   \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn

```

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

```

2396   { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2397 }
2398 {
2399   \@@_create_col_nodes:
2400   \endarray
2401 }

```

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

```

2402 \NewDocumentEnvironment { @@-light-syntax } { b }
2403 {

```

First, we test whether the environment is empty. It's only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in `#1`.

```

2404   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2405   \tl_map_inline:nn { #1 }
2406   {
2407     \str_if_eq:nnT { ##1 } { & }
2408     { \@@_fatal:n { ampersand-in-light-syntax } }
2409     \str_if_eq:nnT { ##1 } { \ }
2410     { \@@_fatal:n { double-backslash-in-light-syntax } }
2411   }

```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

```

2412   \@@_light_syntax_i #1 \CodeAfter \q_stop
2413 }

```

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

```

2414 { }
2415 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2416 {
2417   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

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

```

2418   \seq_gclear_new:N \g_@@_rows_seq
2419   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2420   \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

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

```

2421   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2422     { \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }

```

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

```

2423   \exp_args:NV \@@_array: \g_@@_preamble_tl

```

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

```

2424   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2425   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2426   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2427   \@@_create_col_nodes:
2428   \endarray
2429 }
2430 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2431 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2432 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2433 {
2434   \seq_gclear_new:N \g_@@_cells_seq
2435   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2436   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2437   \l_tmpa_tl
2438   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2439 }

```

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).

```

2440 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2441 {
2442   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2443     { \@@_fatal:n { empty~environment } }

```

We reprint in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```

2444   \end { #2 }
2445 }

```

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).

```

2446 \cs_new:Npn \@@_create_col_nodes:
2447 {
2448   \crrc
2449   \int_compare:nNnT \l_@@_first_col_int = 0
2450     {
2451       \omit

```

```

2452 \hbox_overlap_left:n
2453 {
2454   \bool_if:NT \l_@@_code_before_bool
2455     { \pgfsys@markposition { \@@_env: - col - 0 } }
2456   \pgfpicture
2457     \pgfrememberpicturerepositiononpagetrue
2458     \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
2459     \str_if_empty:NF \l_@@_name_str
2460       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2461     \endpgfpicture
2462     \skip_horizontal:N 2\col@sep
2463     \skip_horizontal:N \g_@@_width_first_col_dim
2464   }
2465   &
2466 }
2467 \omit

```

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

```

2468 \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`).

```

2469 \int_compare:nNnTF \l_@@_first_col_int = 0
2470 {
2471   \bool_if:NT \l_@@_code_before_bool
2472   {
2473     \hbox
2474     {
2475       \skip_horizontal:N -0.5\arrayrulewidth
2476       \pgfsys@markposition { \@@_env: - col - 1 }
2477       \skip_horizontal:N 0.5\arrayrulewidth
2478     }
2479   }
2480   \pgfpicture
2481   \pgfrememberpicturerepositiononpagetrue
2482   \pgfcoordinate { \@@_env: - col - 1 }
2483     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2484   \str_if_empty:NF \l_@@_name_str
2485     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2486   \endpgfpicture
2487 }
2488 {
2489   \bool_if:NT \l_@@_code_before_bool
2490   {
2491     \hbox
2492     {
2493       \skip_horizontal:N 0.5\arrayrulewidth
2494       \pgfsys@markposition { \@@_env: - col - 1 }
2495       \skip_horizontal:N -0.5\arrayrulewidth
2496     }
2497   }
2498   \pgfpicture
2499   \pgfrememberpicturerepositiononpagetrue
2500   \pgfcoordinate { \@@_env: - col - 1 }
2501     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2502   \str_if_empty:NF \l_@@_name_str
2503     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2504   \endpgfpicture
2505 }

```

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

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

```

2506 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
2507 \bool_if:NF \l_@@_auto_columns_width_bool
2508 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2509 {
2510   \bool_lazy_and:nnTF
2511     \l_@@_auto_columns_width_bool
2512     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2513     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2514     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2515     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2516 }
2517 \skip_horizontal:N \g_tmpa_skip
2518 \hbox
2519 {
2520   \bool_if:NT \l_@@_code_before_bool
2521   {
2522     \hbox
2523     {
2524       \skip_horizontal:N -0.5\arrayrulewidth
2525       \pgfsys@markposition { \@@_env: - col - 2 }
2526       \skip_horizontal:N 0.5\arrayrulewidth
2527     }
2528   }
2529   \pgfpicture
2530   \pgfrememberpicturerepositiononpagetrue
2531   \pgfcoordinate { \@@_env: - col - 2 }
2532   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2533   \str_if_empty:NF \l_@@_name_str
2534   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2535   \endpgfpicture
2536 }

```

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

```

2537 \int_gset:Nn \g_tmpa_int 1
2538 \bool_if:NTF \g_@@_last_col_found_bool
2539 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2540 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2541 {
2542   &
2543   \omit
2544   \int_gincr:N \g_tmpa_int

```

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

```

2545 \skip_horizontal:N \g_tmpa_skip
2546 \bool_if:NT \l_@@_code_before_bool
2547 {
2548   \hbox
2549   {
2550     \skip_horizontal:N -0.5\arrayrulewidth
2551     \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2552     \skip_horizontal:N 0.5\arrayrulewidth
2553   }
2554 }

```

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

```

2555 \pgfpicture
2556 \pgfrememberpicturerepositiononpagetrue
2557 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2558 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2559 \str_if_empty:NF \l_@@_name_str
2560 {

```

```

2561         \pgfnodealias
2562         { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2563         { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2564     }
2565 \endpgfpicture
2566 }

```

```

2567 &
2568 \omit
2569 \int_gincr:N \g_tmpa_int
2570 \skip_horizontal:N \g_tmpa_skip
2571 \bool_lazy_all:nT
2572 {
2573     \l_@@_NiceArray_bool
2574     { \bool_not_p:n \l_@@_NiceTabular_bool }
2575     { \clist_if_empty_p:N \l_@@_vlines_clist }
2576     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2577     { ! \l_@@_bar_at_end_of_pream_bool }
2578 }
2579 { \skip_horizontal:N -\col@sep }
2580 \bool_if:NT \l_@@_code_before_bool
2581 {
2582     \hbox
2583     {
2584         \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.

```

2585         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2586         { \skip_horizontal:N -\arraycolsep }
2587         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2588         \skip_horizontal:N 0.5\arrayrulewidth
2589         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2590         { \skip_horizontal:N \arraycolsep }
2591     }
2592 }
2593 \pgfpicture
2594 \pgfrememberpicturepositiononpagetrue
2595 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2596 {
2597     \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2598     {
2599         \pgfpoint
2600         { - 0.5 \arrayrulewidth - \arraycolsep }
2601         \c_zero_dim
2602     }
2603     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2604 }
2605 \str_if_empty:NF \l_@@_name_str
2606 {
2607     \pgfnodealias
2608     { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2609     { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2610 }
2611 \endpgfpicture

2612 \bool_if:NT \g_@@_last_col_found_bool
2613 {
2614     \hbox_overlap_right:n
2615     {
2616         \skip_horizontal:N \g_@@_width_last_col_dim

```

```

2617         \bool_if:NT \l_@@_code_before_bool
2618         {
2619             \pgfsys@markposition
2620             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2621         }
2622         \pgfpicture
2623         \pgfrememberpicturepositiononpagetrue
2624         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2625         \pgfpointorigin
2626         \str_if_empty:NF \l_@@_name_str
2627         {
2628             \pgfnodealias
2629             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2630             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2631         }
2632         \endpgfpicture
2633     }
2634 }
2635 \cr
2636 }

```

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

```

2637 \tl_const:Nn \c_@@_preamble_first_col_tl
2638 {
2639     >
2640     {

```

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

```

2641         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
2642         \bool_gset_true:N \g_@@_after_col_zero_bool
2643         \@@_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.

```

2644         \hbox_set:Nw \l_@@_cell_box
2645         \@@_math_toggle_token:
2646         \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”.

```

2647         \bool_lazy_and:nnT
2648         { \int_compare_p:nNn \c@iRow > 0 }
2649         {
2650             \bool_lazy_or_p:nn
2651             { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2652             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2653         }
2654         {
2655             \l_@@_code_for_first_col_tl
2656             \xglobal \colorlet { nicematrix-first-col } { . }
2657         }
2658     }

```

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`.

```

2659     l
2660     <
2661     {
2662         \@@_math_toggle_token:
2663         \hbox_set_end:
2664         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2665         \@@_adjust_size_box:
2666         \@@_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.

```
2667 \dim_gset:Nn \g_@@_width_first_col_dim
2668 { \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.

```
2669 \hbox_overlap_left:n
2670 {
2671   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2672     \@@_node_for_cell:
2673     { \box_use_drop:N \l_@@_cell_box }
2674     \skip_horizontal:N \l_@@_left_delim_dim
2675     \skip_horizontal:N \l_@@_left_margin_dim
2676     \skip_horizontal:N \l_@@_extra_left_margin_dim
2677   }
2678   \bool_gset_false:N \g_@@_empty_cell_bool
2679   \skip_horizontal:N -2\col@sep
2680 }
2681 }
```

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

```
2682 \tl_const:Nn \c_@@_preamble_last_col_tl
2683 {
2684   >
2685   {
```

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

```
2686 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:n
```

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

```
2687 \bool_gset_true:N \g_@@_last_col_found_bool
2688 \int_gincr:N \c@jCol
2689 \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.

```
2690 \hbox_set:Nw \l_@@_cell_box
2691 \@@_math_toggle_token:
2692 \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”.

```
2693 \int_compare:nNnT \c@iRow > 0
2694 {
2695   \bool_lazy_or:nnT
2696     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2697     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2698   {
2699     \l_@@_code_for_last_col_tl
2700     \xglobal \colorlet { nicematrix-last-col } { . }
2701   }
2702 }
2703 }
2704 1
2705 <
2706 {
2707   \@@_math_toggle_token:
2708   \hbox_set_end:
2709   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2710   \@@_adjust_size_box:
2711   \@@_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.

```

2712 \dim_gset:Nn \g_@@_width_last_col_dim
2713 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2714 \skip_horizontal:N -2\col@sep

```

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

```

2715 \hbox_overlap_right:n
2716 {
2717   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2718   {
2719     \skip_horizontal:N \l_@@_right_delim_dim
2720     \skip_horizontal:N \l_@@_right_margin_dim
2721     \skip_horizontal:N \l_@@_extra_right_margin_dim
2722     \@@_node_for_cell:
2723   }
2724 }
2725 \bool_gset_false:N \g_@@_empty_cell_bool
2726 }
2727 }

```

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.

```

2728 \NewDocumentEnvironment { NiceArray } { }
2729 {
2730   \bool_set_true:N \l_@@_NiceArray_bool
2731   \str_if_empty:NT \g_@@_name_env_str
2732   { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put . and . for the delimiters but, in fact, that doesn’t matter because these arguments won’t be used in {NiceArrayWithDelims} (because the flag \l_@@_NiceArray_bool is raised).

```

2733   \NiceArrayWithDelims . .
2734 }
2735 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

2736 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2737 {
2738   \NewDocumentEnvironment { #1 NiceArray } { }
2739   {
2740     \str_if_empty:NT \g_@@_name_env_str
2741     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2742     \@@_test_if_math_mode:
2743     \NiceArrayWithDelims #2 #3
2744   }
2745   { \endNiceArrayWithDelims }
2746 }
2747 \@@_def_env:nnn p ( )
2748 \@@_def_env:nnn b [ ]
2749 \@@_def_env:nnn B \{ \}
2750 \@@_def_env:nnn v | |
2751 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2752 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2753 {
2754   \bool_set_true:N \l_@@_Matrix_bool
2755   \use:c { #1 NiceArray }
2756   {
2757     *
2758     {
2759       \int_compare:nNnTF \l_@@_last_col_int < 0
2760       \c@MaxMatrixCols
2761       { \@@_pred:n \l_@@_last_col_int }
2762     }
2763     { > \@@_Cell: #2 < \@@_end_Cell: }
2764   }
2765 }
2766 \clist_map_inline:nn { { } , p , b , B , v , V }
2767 {
2768   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
2769   {
2770     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2771     \tl_set:Nn \l_@@_type_of_col_tl c
2772     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2773     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2774   }
2775   { \use:c { end #1 NiceArray } }
2776 }

```

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

```

2777 \cs_new_protected:Npn \@@_NotEmpty:
2778 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

The environments {NiceTabular} and {NiceTabular*}

```

2779 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2780 {
2781   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2782   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2783   \bool_set_true:N \l_@@_NiceTabular_bool
2784   \NiceArray { #2 }
2785 }
2786 { \endNiceArray }

2787 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
2788 {
2789   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
2790   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
2791   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2792   \bool_set_true:N \l_@@_NiceTabular_bool
2793   \NiceArray { #3 }
2794 }
2795 { \endNiceArray }

```

After the construction of the array

```

2796 \cs_new_protected:Npn \@@_after_array:
2797 {
2798   \group_begin:

```

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 `\Ddots` drawn in that last column. That's why we fix the correct value of `\l_@@_last_col_int` in that case.

```
2799 \bool_if:NT \g_@@_last_col_found_bool
2800 { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }
```

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

```
2801 \bool_if:NT \l_@@_last_col_without_value_bool
2802 { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }
```

It's also time to give to `\l_@@_last_row_int` its real value.

```
2803 \bool_if:NT \l_@@_last_row_without_value_bool
2804 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
2805 \tl_gput_right:Nx \g_@@_aux_tl
2806 {
2807   \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
2808   {
2809     \int_use:N \l_@@_first_row_int ,
2810     \int_use:N \c_iRow ,
2811     \int_use:N \g_@@_row_total_int ,
2812     \int_use:N \l_@@_first_col_int ,
2813     \int_use:N \c_jCol ,
2814     \int_use:N \g_@@_col_total_int
2815   }
2816 }
```

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

```
2817 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
2818 {
2819   \tl_gput_right:Nx \g_@@_aux_tl
2820   {
2821     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
2822     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
2823   }
2824 }
2825 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
2826 {
2827   \tl_gput_right:Nx \g_@@_aux_tl
2828   {
2829     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
2830     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
2831     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
2832     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
2833   }
2834 }
```

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

```
2835 \@@_create_diag_nodes:
```

By default, the diagonal lines will be parallelized⁶¹. 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.

```
2836 \bool_if:NT \l_@@_parallelize_diags_bool
2837 {
2838   \int_gzero_new:N \g_@@_ddots_int
2839   \int_gzero_new:N \g_@@_iddots_int
```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots`

⁶¹It's possible to use the option `parallelize-diags` to disable this parallelization.

diagonals parallel to the first one. Similarly Δ_x and Δ_y of the first Δ diagonal.

```

2840 \dim_gzero_new:N \g_@@_delta_x_one_dim
2841 \dim_gzero_new:N \g_@@_delta_y_one_dim
2842 \dim_gzero_new:N \g_@@_delta_x_two_dim
2843 \dim_gzero_new:N \g_@@_delta_y_two_dim
2844 }
2845 \int_zero_new:N \l_@@_initial_i_int
2846 \int_zero_new:N \l_@@_initial_j_int
2847 \int_zero_new:N \l_@@_final_i_int
2848 \int_zero_new:N \l_@@_final_j_int
2849 \bool_set_false:N \l_@@_initial_open_bool
2850 \bool_set_false:N \l_@@_final_open_bool

```

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

```

2851 \bool_if:NT \l_@@_small_bool
2852 {
2853   \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
2854   \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension Δ_x corresponds to the option `xdots/shorten` available to the user. That's why we give a new value according to the current value, and not an absolute value.

```

2855   \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
2856 }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

2857 \@@_draw_dotted_lines:

```

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.

```

2858 \@@_compute_corners:

```

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

```

2859 \@@_adjust_pos_of_blocks_seq:

```

The following code is only for efficiency. We determine whether the potential horizontal and vertical rules are “complete”, that is to say drawn in the whole array. We are sure that all the rules will be complete when there is no block, no virtual block (determined by a command such as `\Cdots`, `\Vdots`, etc.) and no corners. In that case, we switch to a shortcut version of `\@@_vline_i:nn` and `\@@_hline:nn`.

```

2860 \bool_lazy_all:nT
2861 {
2862   { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
2863   { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
2864   { \seq_if_empty_p:N \l_@@_corners_cells_seq }
2865 }
2866 {
2867   \cs_set_eq:NN \@@_vline_i:nn \@@_vline_i_complete:nn
2868   \cs_set_eq:NN \@@_hline_i:nn \@@_hline_i_complete:nn
2869 }
2870 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
2871 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
2872 \cs_set_eq:NN \SubMatrix \@@_SubMatrix

```

Now, the internal code-after and then, the `\CodeAfter`.

```

2873 \bool_if:NT \c_@@_tikz_loaded_bool
2874 {

```



```

2875     \tikzset
2876     {
2877         every~picture / .style =
2878         {
2879             overlay ,
2880             remember~picture ,
2881             name-prefix = \@@_env: -
2882         }
2883     }
2884 }
2885 \cs_set_eq:NN \line \@@_line
2886 \g_@@_internal_code_after_tl
2887 \tl_gclear:N \g_@@_internal_code_after_tl

```

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.

```

2888 \cs_set_eq:NN \CodeAfter \prg_do_nothing:

```

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

```

2889 \seq_gclear:N \g_@@_submatrix_names_seq

```

We compose the `code-after` in math mode in order to nullify the spaces put by the user between instructions in the `code-after`.

```

2890 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

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:`.

```

2891 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
2892 \scan_stop:
2893 % \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
2894 \tl_gclear:N \g_nicematrix_code_after_tl
2895 \group_end:

```

`\g_nicematrix_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor` (when the key `colortbl-like` is in force). These instructions will be written on the `aux` file to be added to the `code-before` in the next run.

```

2896 \tl_if_empty:NF \g_nicematrix_code_before_tl
2897 {

```

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`.

```

2898 \cs_set_protected:Npn \rectanglecolor { }
2899 \cs_set_protected:Npn \columncolor { }
2900 \tl_gput_right:Nx \g_@@_aux_tl
2901 {
2902     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
2903     { \exp_not:V \g_nicematrix_code_before_tl }
2904 }
2905 \bool_set_true:N \l_@@_code_before_bool
2906 }
2907 % \bool_if:NT \l_@@_code_before_bool \@@_write_aux_for_cell_nodes:

2908 \str_gclear:N \g_@@_name_env_str
2909 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶². This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able

⁶²e.g. `\color[rgb]{0.5,0.5,0}`

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`.

```
2910 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
2911 }
```

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, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```
2912 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
2913 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

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

```
2914 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
2915 {
2916   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
2917   { \@@_adjust_pos_of_blocks_seq_i:nnnn ##1 }
2918 }
```

The following command must *not* be protected.

```
2919 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnn #1 #2 #3 #4
2920 {
2921   { #1 }
2922   { #2 }
2923   {
2924     \int_compare:nNnTF { #3 } > { 99 }
2925     { \int_use:N \c@iRow }
2926     { #3 }
2927   }
2928   {
2929     \int_compare:nNnTF { #4 } > { 99 }
2930     { \int_use:N \c@jCol }
2931     { #4 }
2932   }
2933 }
```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we have to define the adequate version of `\@@_draw_dotted_lines`: whether Tikz is loaded or not (in that case, only PGF is loaded).

```
2934 \AtBeginDocument
2935 {
2936   \cs_new_protected:Npx \@@_draw_dotted_lines:
2937   {
2938     \c_@@_pgfortikzpicture_tl
2939     \@@_draw_dotted_lines_i:
2940     \c_@@_endpgfortikzpicture_tl
2941   }
2942 }
```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```
2943 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
2944 {
2945   \pgfrememberpicturepositiononpagetrue
2946   \pgf@relevantforpicturesizefalse
2947   \g_@@_HVdotsfor_lines_tl
2948   \g_@@_Vdots_lines_tl
2949   \g_@@_Ddots_lines_tl
```

```

2950 \g_@@_Iddots_lines_tl
2951 \g_@@_Cdots_lines_tl
2952 \g_@@_Ldots_lines_tl
2953 }

2954 \cs_new_protected:Npn \@@_restore_iRow_jCol:
2955 {
2956   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
2957   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
2958 }

```

We define a new PGF shape for the diag nodes because we want to provide a anchor called .5 for those nodes.

```

2959 \pgfdeclareshape { @@_diag_node }
2960 {
2961   \savedanchor { \five }
2962   {
2963     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
2964     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
2965   }
2966   \anchor { 5 } { \five }
2967   \anchor { center } { \pgfpointorigin }
2968 }

2969 \cs_new_protected:Npn \@@_write_aux_for_cell_nodes:
2970 {
2971   \pgfpicture
2972   \pgfrememberpicturerepositiononpagetrue
2973   \pgf@relevantforpicturesizefalse
2974   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2975   {
2976     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2977     {
2978       \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
2979       {
2980         \pgfscope
2981         \pgftransformshift
2982         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { north-west } }
2983         \pgfnode
2984         { rectangle }
2985         { center }
2986         {
2987           \hbox
2988           { \pgfsys@markposition { \@@_env: - ##1 - #####1 - NW } }
2989         }
2990         { }
2991         { }
2992         \endpgfscope
2993         \pgfscope
2994         \pgftransformshift
2995         { \pgfpointanchor { \@@_env: - ##1 - #####1 } { south-east } }
2996         \pgfnode
2997         { rectangle }
2998         { center }
2999         {
3000           \hbox
3001           { \pgfsys@markposition { \@@_env: - ##1 - #####1 - SE } }
3002         }
3003         { }
3004         { }
3005         \endpgfscope
3006       }

```

```

3007     }
3008   }
3009   \endpgfpicture
3010   \@@_create_extra_nodes:
3011 }
3012 % \end{macrocode}
3013 %
3014 % \bigskip
3015 % The following command creates the diagonal nodes (in fact, if the matrix is
3016 % not a square matrix, not all the nodes are on the diagonal).
3017 % \begin{macrocode}
3018 \cs_new_protected:Npn \@@_create_diag_nodes:
3019 {
3020   \pgfpicture
3021   \pgfrememberpicturepositiononpagetrue
3022   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3023   {
3024     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3025     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3026     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3027     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3028     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3029     \dim_set_eq:NN \l_tmpc_dim \pgf@x
3030     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3031     \dim_set_eq:NN \l_tmpd_dim \pgf@y
3032     \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@a_diag_node`) that we will construct.

```

3033     \dim_set:Nn \l_tmpa_dim { ( \l_tmpc_dim - \l_tmpa_dim ) / 2 }
3034     \dim_set:Nn \l_tmpb_dim { ( \l_tmpd_dim - \l_tmpb_dim ) / 2 }
3035     \pgfnode { @a_diag_node } { center } { } { \@@_env: - ##1 } { }
3036     \str_if_empty:NF \l_@@_name_str
3037     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3038   }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

3039     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3040     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3041     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3042     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3043     \pgfcoordinate
3044     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3045     \pgfnodealias
3046     { \@@_env: - last }
3047     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3048     \str_if_empty:NF \l_@@_name_str
3049     {
3050       \pgfnodealias
3051       { \l_@@_name_str - \int_use:N \l_tmpa_int }
3052       { \@@_env: - \int_use:N \l_tmpa_int }
3053       \pgfnodealias
3054       { \l_@@_name_str - last }
3055       { \@@_env: - last }
3056     }
3057   \endpgfpicture
3058 }

```

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.

$$\begin{pmatrix} a+b+c & a+b & a \\ a & \dots & \dots \\ a & a+b & a+b+c \end{pmatrix}$$

The command `\l_@@_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.

```
3059 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3060 {
```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```
3061 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3062 \int_set:Nn \l_@@_initial_i_int { #1 }
3063 \int_set:Nn \l_@@_initial_j_int { #2 }
3064 \int_set:Nn \l_@@_final_i_int { #1 }
3065 \int_set:Nn \l_@@_final_j_int { #2 }
```

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

```
3066 \bool_set_false:N \l_@@_stop_loop_bool
3067 \bool_do_until:Nn \l_@@_stop_loop_bool
3068 {
3069   \int_add:Nn \l_@@_final_i_int { #3 }
3070   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3071   \bool_set_false:N \l_@@_final_open_bool
3072   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3073   {
3074     \int_compare:nNnTF { #3 } = 1
3075     { \bool_set_true:N \l_@@_final_open_bool }
3076     {
3077       \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3078       { \bool_set_true:N \l_@@_final_open_bool }
3079     }
3080   }
3081   {
3082     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3083     {
3084       \int_compare:nNnTF { #4 } = { -1 }
3085       { \bool_set_true:N \l_@@_final_open_bool }
3086     }
3087     {
```

```

3088         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3089         {
3090             \int_compare:nNtT { #4 } = 1
3091             { \bool_set_true:N \l_@@_final_open_bool }
3092         }
3093     }
3094 }
3095 \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.

```

3096 {

```

We do a step backwards.

```

3097     \int_sub:Nn \l_@@_final_i_int { #3 }
3098     \int_sub:Nn \l_@@_final_j_int { #4 }
3099     \bool_set_true:N \l_@@_stop_loop_bool
3100 }

```

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`.

```

3101 {
3102     \cs_if_exist:cTF
3103     {
3104         @@ _ dotted _
3105         \int_use:N \l_@@_final_i_int -
3106         \int_use:N \l_@@_final_j_int
3107     }
3108     {
3109         \int_sub:Nn \l_@@_final_i_int { #3 }
3110         \int_sub:Nn \l_@@_final_j_int { #4 }
3111         \bool_set_true:N \l_@@_final_open_bool
3112         \bool_set_true:N \l_@@_stop_loop_bool
3113     }
3114     {
3115         \cs_if_exist:cTF
3116         {
3117             pgf @ sh @ ns @ \@@_env:
3118             - \int_use:N \l_@@_final_i_int
3119             - \int_use:N \l_@@_final_j_int
3120         }
3121         { \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.

```

3122     {
3123         \cs_set:cpn
3124         {
3125             @@ _ dotted _
3126             \int_use:N \l_@@_final_i_int -
3127             \int_use:N \l_@@_final_j_int
3128         }
3129         { }
3130     }
3131 }
3132 }
3133 }

```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```

3134     \bool_set_false:N \l_@@_stop_loop_bool

```

```

3135 \bool_do_until:Nn \l_@@_stop_loop_bool
3136 {
3137   \int_sub:Nn \l_@@_initial_i_int { #3 }
3138   \int_sub:Nn \l_@@_initial_j_int { #4 }
3139   \bool_set_false:N \l_@@_initial_open_bool
3140   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3141   {
3142     \int_compare:nNnTF { #3 } = 1
3143     { \bool_set_true:N \l_@@_initial_open_bool }
3144     {
3145       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3146       { \bool_set_true:N \l_@@_initial_open_bool }
3147     }
3148   }
3149   {
3150     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3151     {
3152       \int_compare:nNnT { #4 } = 1
3153       { \bool_set_true:N \l_@@_initial_open_bool }
3154     }
3155     {
3156       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3157       {
3158         \int_compare:nNnT { #4 } = { -1 }
3159         { \bool_set_true:N \l_@@_initial_open_bool }
3160       }
3161     }
3162   }
3163   \bool_if:NnTF \l_@@_initial_open_bool
3164   {
3165     \int_add:Nn \l_@@_initial_i_int { #3 }
3166     \int_add:Nn \l_@@_initial_j_int { #4 }
3167     \bool_set_true:N \l_@@_stop_loop_bool
3168   }
3169   {
3170     \cs_if_exist:cTF
3171     {
3172       @@ _ dotted _
3173       \int_use:N \l_@@_initial_i_int -
3174       \int_use:N \l_@@_initial_j_int
3175     }
3176     {
3177       \int_add:Nn \l_@@_initial_i_int { #3 }
3178       \int_add:Nn \l_@@_initial_j_int { #4 }
3179       \bool_set_true:N \l_@@_initial_open_bool
3180       \bool_set_true:N \l_@@_stop_loop_bool
3181     }
3182     {
3183       \cs_if_exist:cTF
3184       {
3185         pgf @ sh @ ns @ \@@_env:
3186         - \int_use:N \l_@@_initial_i_int
3187         - \int_use:N \l_@@_initial_j_int
3188       }
3189       { \bool_set_true:N \l_@@_stop_loop_bool }
3190       {
3191         \cs_set:cpn
3192         {
3193           @@ _ dotted _
3194           \int_use:N \l_@@_initial_i_int -
3195           \int_use:N \l_@@_initial_j_int
3196         }
3197         { }

```

```

3198         }
3199     }
3200 }
3201 }

```

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.

```

3202 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3203 {
3204     { \int_use:N \l_@@_initial_i_int }
3205     { \int_use:N \l_@@_initial_j_int }
3206     { \int_use:N \l_@@_final_i_int }
3207     { \int_use:N \l_@@_final_j_int }
3208 }
3209 }

```

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

```

3210 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3211 {
3212     \int_set:Nn \l_@@_row_min_int 1
3213     \int_set:Nn \l_@@_col_min_int 1
3214     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3215     \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the code-before. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

3216 \seq_map_inline:Nn \g_@@_submatrix_seq
3217 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
3218 }

```

#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.

```

3219 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3220 {
3221     \bool_if:nT
3222     {
3223         \int_compare_p:n { #3 <= #1 }
3224         && \int_compare_p:n { #1 <= #5 }
3225         && \int_compare_p:n { #4 <= #2 }
3226         && \int_compare_p:n { #2 <= #6 }
3227     }
3228     {
3229         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3230         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3231         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3232         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3233     }
3234 }

```

```

3235 \cs_new_protected:Npn \@@_set_initial_coords:
3236 {
3237     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3238     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3239 }
3240 \cs_new_protected:Npn \@@_set_final_coords:
3241 {
3242     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3243     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3244 }

```



```

3245 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3246 {
3247   \pgfpointanchor
3248   {
3249     \@@_env:
3250     - \int_use:N \l_@@_initial_i_int
3251     - \int_use:N \l_@@_initial_j_int
3252   }
3253   { #1 }
3254   \@@_set_initial_coords:
3255 }
3256 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3257 {
3258   \pgfpointanchor
3259   {
3260     \@@_env:
3261     - \int_use:N \l_@@_final_i_int
3262     - \int_use:N \l_@@_final_j_int
3263   }
3264   { #1 }
3265   \@@_set_final_coords:
3266 }
3267 \cs_new_protected:Npn \@@_open_x_initial_dim:
3268 {
3269   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3270   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3271   {
3272     \cs_if_exist:cT
3273     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3274     {
3275       \pgfpointanchor
3276       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3277       { west }
3278       \dim_set:Nn \l_@@_x_initial_dim
3279       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3280     }
3281   }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

3282   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3283   {
3284     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3285     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3286     \dim_add:Nn \l_@@_x_initial_dim \col@sep
3287   }
3288 }
3289 \cs_new_protected:Npn \@@_open_x_final_dim:
3290 {
3291   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3292   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3293   {
3294     \cs_if_exist:cT
3295     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3296     {
3297       \pgfpointanchor
3298       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3299       { east }
3300       \dim_set:Nn \l_@@_x_final_dim
3301       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3302     }
3303   }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

3304 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3305 {
3306   \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3307   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3308   \dim_sub:Nn \l_@@_x_final_dim \col@sep
3309 }
3310 }

```

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.

```

3311 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3312 {
3313   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3314   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3315   {
3316     \@@_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.

```

3317   \group_begin:
3318     \int_compare:nNnTF { #1 } = 0
3319     { \color { nicematrix-first-row } }
3320     {

```

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

```

3321       \int_compare:nNnT { #1 } = \l_@@_last_row_int
3322       { \color { nicematrix-last-row } }
3323     }
3324     \keys_set:nn { NiceMatrix / xdots } { #3 }
3325     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3326     \@@_actually_draw_Ldots:
3327   \group_end:
3328 }
3329 }

```

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`.

```

3330 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3331 {
3332   \bool_if:NTF \l_@@_initial_open_bool
3333   {
3334     \@@_open_x_initial_dim:
3335     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3336     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3337   }
3338   { \@@_set_initial_coords_from_anchor:n { base-east } }
3339   \bool_if:NTF \l_@@_final_open_bool
3340   {
3341     \@@_open_x_final_dim:
3342     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }

```

```

3343     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3344   }
3345   { \@@_set_final_coords_from_anchor:n { base~west } }

```

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

```

3346     \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3347     \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3348     \@@_draw_line:
3349   }

```

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.

```

3350 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3351 {
3352   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3353   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3354   {
3355     \@@_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.

```

3356     \group_begin:
3357     \int_compare:nNnTF { #1 } = 0
3358       { \color { nicematrix-first-row } }
3359     {

```

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

```

3360       \int_compare:nNnT { #1 } = \l_@@_last_row_int
3361       { \color { nicematrix-last-row } }
3362     }
3363     \keys_set:nn { NiceMatrix / xdots } { #3 }
3364     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3365     \@@_actually_draw_Cdots:
3366   \group_end:
3367 }
3368 }

```

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`.

```

3369 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3370 {
3371   \bool_if:NTF \l_@@_initial_open_bool
3372     { \@@_open_x_initial_dim: }
3373     { \@@_set_initial_coords_from_anchor:n { mid~east } }
3374   \bool_if:NTF \l_@@_final_open_bool
3375     { \@@_open_x_final_dim: }
3376     { \@@_set_final_coords_from_anchor:n { mid~west } }
3377   \bool_lazy_and:nnTF
3378     \l_@@_initial_open_bool
3379     \l_@@_final_open_bool
3380   {

```

```

3381 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3382 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3383 \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3384 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3385 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3386 }
3387 {
3388 \bool_if:NT \l_@@_initial_open_bool
3389 { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3390 \bool_if:NT \l_@@_final_open_bool
3391 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3392 }
3393 \@@_draw_line:
3394 }
3395 \cs_new_protected:Npn \@@_open_y_initial_dim:
3396 {
3397 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3398 \dim_set:Nn \l_@@_y_initial_dim
3399 { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3400 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3401 {
3402 \cs_if_exist:cT
3403 { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3404 {
3405 \pgfpointanchor
3406 { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3407 { north }
3408 \dim_set:Nn \l_@@_y_initial_dim
3409 { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3410 }
3411 }
3412 }
3413 \cs_new_protected:Npn \@@_open_y_final_dim:
3414 {
3415 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3416 \dim_set:Nn \l_@@_y_final_dim
3417 { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3418 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3419 {
3420 \cs_if_exist:cT
3421 { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3422 {
3423 \pgfpointanchor
3424 { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3425 { south }
3426 \dim_set:Nn \l_@@_y_final_dim
3427 { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3428 }
3429 }
3430 }

```

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.

```

3431 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3432 {
3433 \@@_adjust_to_submatrix:nn { #1 } { #2 }
3434 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3435 {
3436 \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 0

```

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

```

3437 \group_begin:

```

```

3438     \int_compare:nNnTF { #2 } = 0
3439     { \color { nicematrix-first-col } }
3440     {
3441         \int_compare:nNnT { #2 } = \l_@@_last_col_int
3442         { \color { nicematrix-last-col } }
3443     }
3444     \keys_set:nn { NiceMatrix / xdots } { #3 }
3445     \tl_if_empty:VF \l_@@_xdots_color_tl
3446     { \color { \l_@@_xdots_color_tl } }
3447     \@@_actually_draw_Vdots:
3448 \group_end:
3449 }
3450 }

```

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`.

```

3451 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3452 {

```

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

```

3453     \bool_set_false:N \l_tmpa_bool

```

First the case when the line is closed on both ends.

```

3454     \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3455     {
3456         \@@_set_initial_coords_from_anchor:n { south-west }
3457         \@@_set_final_coords_from_anchor:n { north-west }
3458         \bool_set:Nn \l_tmpa_bool
3459         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3460     }

```

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

```

3461     \bool_if:NTF \l_@@_initial_open_bool
3462     \@@_open_y_initial_dim:
3463     { \@@_set_initial_coords_from_anchor:n { south } }
3464     \bool_if:NTF \l_@@_final_open_bool
3465     \@@_open_y_final_dim:
3466     { \@@_set_final_coords_from_anchor:n { north } }
3467     \bool_if:NTF \l_@@_initial_open_bool
3468     {
3469         \bool_if:NTF \l_@@_final_open_bool
3470         {
3471             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3472             \dim_set_eq:NN \l_tmpa_dim \pgf@x
3473             \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3474             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3475             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim

```

We may think that the final user won't use a "last column" which contains only a command `\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.

```

3476     \int_compare:nNnT \l_@@_last_col_int > { -2 }
3477     {

```

```

3478         \int_compare:nNt \l_@@_initial_j_int = \g_@@_col_total_int
3479         {
3480             \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3481             \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3482             \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3483             \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3484         }
3485     }
3486 }
3487 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3488 }
3489 {
3490     \bool_if:NTF \l_@@_final_open_bool
3491     { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3492     {

```

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

```

3493         \dim_compare:nNf \l_@@_x_initial_dim = \l_@@_x_final_dim
3494         {
3495             \dim_set:Nn \l_@@_x_initial_dim
3496             {
3497                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3498                 \l_@@_x_initial_dim \l_@@_x_final_dim
3499             }
3500             \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3501         }
3502     }
3503 }
3504 \@@_draw_line:
3505 }

```

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.

```

3506 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3507 {
3508     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3509     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3510     {
3511         \@@_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.

```

3512     \group_begin:
3513     \keys_set:nn { NiceMatrix / xdots } { #3 }
3514     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3515     \@@_actually_draw_Ddots:
3516     \group_end:
3517 }
3518 }

```

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

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`

- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

3519 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3520 {
3521   \bool_if:NTF \l_@@_initial_open_bool
3522   {
3523     \@@_open_y_initial_dim:
3524     % \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3525     % \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3526     \@@_open_x_initial_dim:
3527   }
3528   { \@@_set_initial_coords_from_anchor:n { south-east } }
3529   \bool_if:NTF \l_@@_final_open_bool
3530   {
3531     % \@@_open_y_final_dim:
3532     % \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3533     \@@_open_x_final_dim:
3534     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3535   }
3536   { \@@_set_final_coords_from_anchor:n { north-west } }

```

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.

```

3537   \bool_if:NT \l_@@_parallelize_diags_bool
3538   {
3539     \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).

```

3540     \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 Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

3541     {
3542       \dim_gset:Nn \g_@@_delta_x_one_dim
3543       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3544       \dim_gset:Nn \g_@@_delta_y_one_dim
3545       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3546     }

```

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate `\l_@@_x_initial_dim`.

```

3547     {
3548       \dim_set:Nn \l_@@_y_final_dim
3549       {
3550         \l_@@_y_initial_dim +
3551         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3552         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3553       }
3554     }
3555   }
3556   \@@_draw_line:
3557 }

```

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.

```

3558 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3559 {
3560   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3561   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3562   {
3563     \@@_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.

```

3564     \group_begin:
3565     \keys_set:nn { NiceMatrix / xdots } { #3 }
3566     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3567     \@@_actually_draw_Iddots:
3568     \group_end:
3569   }
3570 }

```

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`.

```

3571 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3572 {
3573   \bool_if:NTF \l_@@_initial_open_bool
3574   {
3575     \@@_open_y_initial_dim:
3576     \@@_open_x_initial_dim:
3577   }
3578   { \@@_set_initial_coords_from_anchor:n { south-west } }
3579   \bool_if:NTF \l_@@_final_open_bool
3580   {
3581     \@@_open_y_final_dim:
3582     \@@_open_x_final_dim:
3583   }
3584   { \@@_set_final_coords_from_anchor:n { north-east } }
3585   \bool_if:NT \l_@@_parallelize_diags_bool
3586   {
3587     \int_gincr:N \g_@@_iddots_int
3588     \int_compare:nNnTF \g_@@_iddots_int = 1
3589     {
3590       \dim_gset:Nn \g_@@_delta_x_two_dim
3591       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3592       \dim_gset:Nn \g_@@_delta_y_two_dim
3593       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3594     }
3595     {
3596       \dim_set:Nn \l_@@_y_final_dim
3597       {
3598         \l_@@_y_initial_dim +
3599         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3600         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3601       }
3602     }
3603   }
3604   \@@_draw_line:
3605 }

```


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`

```

3606 \cs_new_protected:Npn \@@_draw_line:
3607 {
3608   \pgfrememberpicturepositiononpagetrue
3609   \pgf@relevantforpicturesizefalse
3610   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
3611     \@@_draw_standard_dotted_line:
3612     \@@_draw_non_standard_dotted_line:
3613 }

```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```

3614 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
3615 {
3616   \begin { scope }
3617   \exp_args:No \@@_draw_non_standard_dotted_line:n
3618     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3619 }

```

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

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

```

3620 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
3621 {
3622   \@@_draw_non_standard_dotted_line:nVV
3623     { #1 }
3624     \l_@@_xdots_up_tl
3625     \l_@@_xdots_down_tl
3626 }
3627 \cs_new_protected:Npn \@@_draw_non_standard_dotted_line:nnn #1 #2 #3
3628 {
3629   \draw
3630   [
3631     #1 ,
3632     shorten~> = \l_@@_xdots_shorten_dim ,
3633     shorten~< = \l_@@_xdots_shorten_dim ,
3634   ]
3635     ( \l_@@_x_initial_dim , \l_@@_y_initial_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).

```

3636   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3637   node [ sloped , below ] { $ \scriptstyle #3 $ }
3638   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3639 \end { scope }
3640 }
3641 \cs_generate_variant:Nn \@@_draw_non_standard_dotted_line:nnn { n V V }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real round dots).

```

3642 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3643 {
3644   \bool_lazy_and:nnF
3645     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3646     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3647   {
3648     \pgfscope
3649     \pgftransformshift
3650       {
3651         \pgfpointlineattime { 0.5 }
3652         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3653         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3654       }
3655     \pgftransformrotate
3656       {
3657         \fp_eval:n
3658           {
3659             atand
3660             (
3661               \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3662               \l_@@_x_final_dim - \l_@@_x_initial_dim
3663             )
3664           }
3665       }
3666     \pgfnode
3667       { rectangle }
3668       { south }
3669       {
3670         \c_math_toggle_token
3671         \scriptstyle \l_@@_xdots_up_tl
3672         \c_math_toggle_token
3673       }
3674       { }
3675       { \pgfusepath { } }
3676     \pgfnode
3677       { rectangle }
3678       { north }
3679       {
3680         \c_math_toggle_token
3681         \scriptstyle \l_@@_xdots_down_tl
3682         \c_math_toggle_token
3683       }
3684       { }
3685       { \pgfusepath { } }
3686     \endpgfscope
3687   }
3688   \group_begin:

```

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.

```

3689   \dim_zero_new:N \l_@@_l_dim
3690   \dim_set:Nn \l_@@_l_dim
3691     {
3692       \fp_to_dim:n
3693         {
3694           sqrt
3695             (
3696               ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3697               +
3698               ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3699             )
3700         }

```

3701 }

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.

```

3702        \bool_lazy_or:nnF
3703        { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3704        { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3705        \@@_draw_standard_dotted_line_i:
3706    \group_end:
3707    }
3708    \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3709    \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3710    {

```

The number of dots will be `\l_tmpa_int + 1`.

```

3711        \bool_if:NTF \l_@@_initial_open_bool
3712        {
3713        \bool_if:NTF \l_@@_final_open_bool
3714        {
3715        \int_set:Nn \l_tmpa_int
3716        { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3717        }
3718        {
3719        \int_set:Nn \l_tmpa_int
3720        {
3721        \dim_ratio:nn
3722        { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3723        \l_@@_inter_dots_dim
3724        }
3725        }
3726        }
3727        {
3728        \bool_if:NTF \l_@@_final_open_bool
3729        {
3730        \int_set:Nn \l_tmpa_int
3731        {
3732        \dim_ratio:nn
3733        { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3734        \l_@@_inter_dots_dim
3735        }
3736        }
3737        {
3738        \int_set:Nn \l_tmpa_int
3739        {
3740        \dim_ratio:nn
3741        { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3742        \l_@@_inter_dots_dim
3743        }
3744        }
3745        }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

3746        \dim_set:Nn \l_tmpa_dim
3747        {
3748        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3749        \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3750        }
3751        \dim_set:Nn \l_tmpb_dim
3752        {
3753        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

3754     \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3755 }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in `\l_tmpb_int`.

```

3756 \int_set:Nn \l_tmpb_int
3757 {
3758   \bool_if:NTF \l_@@_initial_open_bool
3759   { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3760   { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3761 }

```

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.

```

3762 \dim_gadd:Nn \l_@@_x_initial_dim
3763 {
3764   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3765   \dim_ratio:nn
3766   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3767   { 2 \l_@@_l_dim }
3768   * \l_tmpb_int
3769 }
3770 \dim_gadd:Nn \l_@@_y_initial_dim
3771 {
3772   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3773   \dim_ratio:nn
3774   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3775   { 2 \l_@@_l_dim }
3776   * \l_tmpb_int
3777 }
3778 \pgf@relevantforpicturesizefalse
3779 \int_step_inline:nnn 0 \l_tmpa_int
3780 {
3781   \pgfpathcircle
3782   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3783   { \l_@@_radius_dim }
3784   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3785   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3786 }
3787 \pgfusepathqfill
3788 }

```

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 that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```

3789 \AtBeginDocument
3790 {
3791   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3792   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

3793 \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3794 {
3795   \int_compare:nNnTF \c@jCol = 0
3796   { \@@_error:nn { in~first~col } \Ldots }
3797   {
3798     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3799     { \@@_error:nn { in~last~col } \Ldots }
3800     {
3801       \@@_instruction_of_type:nnn \c_false_bool { Ldots }
3802       { #1 , down = #2 , up = #3 }
3803     }
3804   }
3805   \bool_if:NF \l_@@_nullify_dots_bool
3806   { \phantom { \ensuremath { \@@_old_ldots } } }
3807   \bool_gset_true:N \g_@@_empty_cell_bool
3808 }

3809 \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3810 {
3811   \int_compare:nNnTF \c@jCol = 0
3812   { \@@_error:nn { in~first~col } \Cdots }
3813   {
3814     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3815     { \@@_error:nn { in~last~col } \Cdots }
3816     {
3817       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
3818       { #1 , down = #2 , up = #3 }
3819     }
3820   }
3821   \bool_if:NF \l_@@_nullify_dots_bool
3822   { \phantom { \ensuremath { \@@_old_cdots } } }
3823   \bool_gset_true:N \g_@@_empty_cell_bool
3824 }

3825 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
3826 {
3827   \int_compare:nNnTF \c@iRow = 0
3828   { \@@_error:nn { in~first~row } \Vdots }
3829   {
3830     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
3831     { \@@_error:nn { in~last~row } \Vdots }
3832     {
3833       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
3834       { #1 , down = #2 , up = #3 }
3835     }
3836   }
3837   \bool_if:NF \l_@@_nullify_dots_bool
3838   { \phantom { \ensuremath { \@@_old_vdots } } }
3839   \bool_gset_true:N \g_@@_empty_cell_bool
3840 }

3841 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
3842 {
3843   \int_case:nnF \c@iRow
3844   {
3845     0 { \@@_error:nn { in~first~row } \Ddots }
3846     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
3847   }
3848   {
3849     \int_case:nnF \c@jCol
3850     {

```

```

3851         0 { \@@_error:nn { in~first~col } \Ddots }
3852         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
3853     }
3854     {
3855         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
3856         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
3857         { #1 , down = #2 , up = #3 }
3858     }
3859
3860 }
3861 \bool_if:NF \l_@@_nullify_dots_bool
3862 { \phantom { \ensuremath { \@@_old_ddots } } }
3863 \bool_gset_true:N \g_@@_empty_cell_bool
3864 }

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
{
\int_case:nnF \c@iRow
{
0 { \@@_error:nn { in~first~row } \Iddots }
\l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
}
{
\int_case:nnF \c@jCol
{
0 { \@@_error:nn { in~first~col } \Iddots }
\l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
}
{
\keys_set_known:nn { NiceMatrix / Ddots } { #1 }
\@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
{ #1 , down = #2 , up = #3 }
}
}
\bool_if:NF \l_@@_nullify_dots_bool
{ \phantom { \ensuremath { \@@_old_iddots } } }
\bool_gset_true:N \g_@@_empty_cell_bool
}
}

```

End of the \AtBeginDocument.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

3889 \keys_define:nn { NiceMatrix / Ddots }
3890 {
3891     draw-first .bool_set:N = \l_@@_draw_first_bool ,
3892     draw-first .default:n = true ,
3893     draw-first .value_forbidden:n = true
3894 }

```

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

```

3895 \cs_new_protected:Npn \@@_Hspace:
3896 {
3897     \bool_gset_true:N \g_@@_empty_cell_bool
3898     \hspace
3899 }
3900 \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`.

```

3901 \cs_new:Npn \@@_Hdotsfor:
3902 {
3903   \bool_lazy_and:nnTF
3904     { \int_compare_p:nNn \c@jCol = 0 }
3905     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
3906     {
3907       \bool_if:NTF \g_@@_after_col_zero_bool
3908       {
3909         \multicolumn { 1 } { c } { }
3910         \@@_Hdotsfor_i
3911       }
3912       { \@@_fatal:n { Hdotsfor~in~col~0 } }
3913     }
3914     {
3915       \multicolumn { 1 } { c } { }
3916       \@@_Hdotsfor_i
3917     }
3918 }

```

The command `\@@_Hdotsfor_i` is defined with `\NewDocumentCommand` because it has an optional argument. Note that such a command defined by `\NewDocumentCommand` is protected and that's why we have put the `\multicolumn` before (in the definition of `\@@_Hdotsfor:`).

```

3919 \AtBeginDocument
3920 {
3921   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3922   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

3923   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
3924   {
3925     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
3926     {
3927       \@@_Hdotsfor:nnnn
3928       { \int_use:N \c@iRow }
3929       { \int_use:N \c@jCol }
3930       { #2 }
3931       {
3932         #1 , #3 ,
3933         down = \exp_not:n { #4 } ,
3934         up = \exp_not:n { #5 }
3935       }
3936     }
3937     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
3938   }
3939 }

```

Enf of `\AtBeginDocument`.

```

3940 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
3941 {
3942   \bool_set_false:N \l_@@_initial_open_bool
3943   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

3944   \int_set:Nn \l_@@_initial_i_int { #1 }
3945   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

3946   \int_compare:nNnTF { #2 } = 1
3947   {
3948     \int_set:Nn \l_@@_initial_j_int 1
3949     \bool_set_true:N \l_@@_initial_open_bool
3950   }

```

```

3951 {
3952   \cs_if_exist:cTF
3953   {
3954     pgf @ sh @ ns @ \@@_env:
3955     - \int_use:N \l_@@_initial_i_int
3956     - \int_eval:n { #2 - 1 }
3957   }
3958   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
3959   {
3960     \int_set:Nn \l_@@_initial_j_int { #2 }
3961     \bool_set_true:N \l_@@_initial_open_bool
3962   }
3963 }
3964 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
3965 {
3966   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3967   \bool_set_true:N \l_@@_final_open_bool
3968 }
3969 {
3970   \cs_if_exist:cTF
3971   {
3972     pgf @ sh @ ns @ \@@_env:
3973     - \int_use:N \l_@@_final_i_int
3974     - \int_eval:n { #2 + #3 }
3975   }
3976   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
3977   {
3978     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
3979     \bool_set_true:N \l_@@_final_open_bool
3980   }
3981 }
3982 \group_begin:
3983 \int_compare:nNnTF { #1 } = 0
3984 { \color { nicematrix-first-row } }
3985 {
3986   \int_compare:nNnT { #1 } = \g_@@_row_total_int
3987   { \color { nicematrix-last-row } }
3988 }
3989 \keys_set:nn { NiceMatrix / xdots } { #4 }
3990 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3991 \@@_actually_draw_Ldots:
3992 \group_end:

```

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).

```

3993   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
3994   { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
3995 }

```

```

3996 \AtBeginDocument
3997 {
3998   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
3999   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4000   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4001   {
4002     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4003     {
4004       \@@_Vdotsfor:nnnn
4005       { \int_use:N \c@iRow }
4006       { \int_use:N \c@jCol }
4007       { #2 }

```



```

4008         {
4009             #1 , #3 ,
4010             down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4011         }
4012     }
4013 }
4014 }

```

Enf of \AtBeginDocument.

```

4015 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4016 {
4017     \bool_set_false:N \l_@@_initial_open_bool
4018     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

4019     \int_set:Nn \l_@@_initial_j_int { #2 }
4020     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

4021     \int_compare:nNnTF #1 = 1
4022     {
4023         \int_set:Nn \l_@@_initial_i_int 1
4024         \bool_set_true:N \l_@@_initial_open_bool
4025     }
4026     {
4027         \cs_if_exist:cTF
4028         {
4029             pgf @ sh @ ns @ \@@_env:
4030             - \int_eval:n { #1 - 1 }
4031             - \int_use:N \l_@@_initial_j_int
4032         }
4033         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4034         {
4035             \int_set:Nn \l_@@_initial_i_int { #1 }
4036             \bool_set_true:N \l_@@_initial_open_bool
4037         }
4038     }
4039     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4040     {
4041         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4042         \bool_set_true:N \l_@@_final_open_bool
4043     }
4044     {
4045         \cs_if_exist:cTF
4046         {
4047             pgf @ sh @ ns @ \@@_env:
4048             - \int_eval:n { #1 + #3 }
4049             - \int_use:N \l_@@_final_j_int
4050         }
4051         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4052         {
4053             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4054             \bool_set_true:N \l_@@_final_open_bool
4055         }
4056     }
4057     \group_begin:
4058     \int_compare:nNnTF { #2 } = 0
4059     { \color { nicematrix-first-col } }
4060     {
4061         \int_compare:nNnT { #2 } = \g_@@_col_total_int
4062         { \color { nicematrix-last-col } }
4063     }
4064     \keys_set:nn { NiceMatrix / xdots } { #4 }
4065     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```

```

4066 \@@_actually_draw_Vdots:
4067 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

4068 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4069 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4070 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

4071 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells.

First, we write a command with an argument of the format *i-j* and applies the command `\int_eval:n` to *i* and *j* ; this must *not* be protected (and is, of course fully expandable).⁶³

```

4072 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4073 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4074 \AtBeginDocument
4075 {
4076   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4077   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4078   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4079   {
4080     \group_begin:
4081     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4082     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4083     \use:e
4084     {
4085       \@@_line_i:nn
4086       { \@@_double_int_eval:n #2 \q_stop }
4087       { \@@_double_int_eval:n #3 \q_stop }
4088     }
4089     \group_end:
4090   }
4091 }

4092 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4093 {
4094   \bool_set_false:N \l_@@_initial_open_bool
4095   \bool_set_false:N \l_@@_final_open_bool
4096   \bool_if:nTF
4097   {
4098     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4099     ||
4100     \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }

```

⁶³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

4101     }
4102     {
4103         \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4104     }
4105     { \@@_draw_line_ii:nn { #1 } { #2 } }
4106 }
4107 \AtBeginDocument
4108 {
4109     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4110     {

```

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:`.

```

4111         \c_@@_pgfortikzpicture_tl
4112         \@@_draw_line_iii:nn { #1 } { #2 }
4113         \c_@@_endpgfortikzpicture_tl
4114     }
4115 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

4116 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4117 {
4118     \pgfrememberpicturepositiononpagetrue
4119     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4120     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4121     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4122     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4123     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4124     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4125     \@@_draw_line:
4126 }

```

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

Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (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 `fill`—coded `f`— in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor` (which are linked to `\rowcolor`, `\columncolor` and `\rectanglecolor` before the execution of the `code-before`) 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: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *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 `\@@_rowcolor:n`, `\@@_columncolor:n` and `\@@_rectanglecolor:nn` (corresponding of `\@@_rowcolor`, `\@@_columncolor` and `\@@_rectanglecolor`).

`bigskip #1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_color_seq` doesn’t only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```

4127 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4128 {

```

Firt, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```

4129 \int_zero:N \l_tmpa_int
4130 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4131 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4132 \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

4133 {
4134   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4135   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4136 }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

4137 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4138 }
4139 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

4140 \cs_new_protected:Npn \@@_actually_color:
4141 {
4142   \pgfpicture
4143   \pgf@relevantforpicturesizefalse
4144   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4145   {
4146     \color ##2
4147     \use:c { g_@@_color _ ##1 _tl }
4148     \tl_gclear:c { g_@@_color _ ##1 _tl }
4149     \pgfusepath { fill }
4150   }
4151   \endpgfpicture
4152 }
4153 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
4154 {
4155   \tl_set:Nn \l_tmpa_tl { #1 }
4156   \tl_set:Nn \l_tmpb_tl { #2 }
4157 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

4158 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4159 {
4160   \tl_if_blank:nF { #2 }
4161   {
4162     \@@_add_to_colors_seq:xn
4163     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4164     { \@@_rowcolor:n { #3 } }
4165   }
4166 }
4167 \cs_new_protected:Npn \@@_rowcolor:n #1
4168 {
4169   \tl_set:Nn \l_@@_rows_tl { #1 }
4170   \tl_set:Nn \l_@@_cols_tl { - }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4171 \@@_cartesian_path:
4172 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

4173 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4174 {
4175   \tl_if_blank:nF { #2 }
4176   {
4177     \@@_add_to_colors_seq:xn
4178     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4179     { \@@_columncolor:n { #3 } }
4180   }
4181 }
4182 \cs_new_protected:Npn \@@_columncolor:n #1
4183 {
4184   \tl_set:Nn \l_@@_rows_tl { - }
4185   \tl_set:Nn \l_@@_cols_tl { #1 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4186   \@@_cartesian_path:
4187 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

4188 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4189 {
4190   \tl_if_blank:nF { #2 }
4191   {
4192     \@@_add_to_colors_seq:xn
4193     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4194     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4195   }
4196 }

```

The last argument is the radius of the corners of the rectangle.

```

4197 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4198 {
4199   \tl_if_blank:nF { #2 }
4200   {
4201     \@@_add_to_colors_seq:xn
4202     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4203     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4204   }
4205 }

```

The last argument is the radius of the corners of the rectangle.

```

4206 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4207 {
4208   \@@_cut_on_hyphen:w #1 \q_stop
4209   \tl_clear_new:N \l_tmpc_tl
4210   \tl_clear_new:N \l_tmpd_tl
4211   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
4212   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
4213   \@@_cut_on_hyphen:w #2 \q_stop
4214   \tl_set:Nx \l_@@_rows_tl { \l_tmpc_tl - \l_tmpa_tl }
4215   \tl_set:Nx \l_@@_cols_tl { \l_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4216   \@@_cartesian_path:n { #3 }
4217 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
4218 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4219 {
4220   \clist_map_inline:nn { #3 }
4221     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4222 }
```

```
4223 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4224 {
4225   \int_step_inline:nn { \int_use:N \c@iRow }
4226     {
4227       \int_step_inline:nn { \int_use:N \c@jCol }
4228         {
4229           \int_if_even:nTF { ####1 + ##1 }
4230             { \@@_cellcolor [ #1 ] { #2 } }
4231             { \@@_cellcolor [ #1 ] { #3 } }
4232           { ##1 - ####1 }
4233         }
4234     }
4235 }
```

```
4236 \keys_define:nn { NiceMatrix / arraycolor }
4237 { except-corners .code:n = \@@_error:n { key except-corners } }
```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns). The third argument is a optional argument which a list of pairs key-value.

```
4238 \NewDocumentCommand \@@_arraycolor { 0 { } m 0 { } }
4239 {
4240   \keys_set:nn { NiceMatrix / arraycolor } { #3 }
4241   \@@_rectanglecolor [ #1 ] { #2 }
4242     { 1 - 1 }
4243     { \int_use:N \c@iRow - \int_use:N \c@jCol }
4244 }
```

```
4245 \keys_define:nn { NiceMatrix / rowcolors }
4246 {
4247   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4248   respect-blocks .default:n = true ,
4249   cols .tl_set:N = \l_@@_cols_tl ,
4250   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4251   restart .default:n = true ,
4252   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4253 }
```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`. Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

#1 (optional) is the color space ; **#2** is a list of intervals of rows ; **#3** is the first color ; **#4** is the second color ; **#5** is for the optional list of pairs key-value.

```
4254 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4255 {
```

The group is for the options.

```
4256   \group_begin:
4257   \tl_clear_new:N \l_@@_cols_tl
4258   \tl_set:Nn \l_@@_cols_tl { - }
4259   \keys_set:nn { NiceMatrix / rowcolors } { #5 }
```

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

```

4260 \bool_set_true:N \l_tmpa_bool
4261 \bool_if:NT \l_@@_respect_blocks_bool
4262 {

```

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`).

```

4263 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4264 \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
4265 { \@@_not_in_exterior_p:nnnn ##1 }
4266 }
4267 \pgfpicture
4268 \pgf@relevantforpicturesizefalse
4269 \clist_map_inline:nn { #2 }
4270 {
4271 \tl_set:Nn \l_tmpa_tl { ##1 }
4272 \tl_if_in:NnTF \l_tmpa_tl { - }
4273 { \@@_cut_on_hyphen:w ##1 \q_stop }
4274 { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

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

```

4275 \int_set:Nn \l_tmpa_int \l_tmpa_tl
4276 \bool_if:NTF \l_@@_rowcolors_restart_bool
4277 { \bool_set_true:N \l_tmpa_bool }
4278 { \bool_set:Nn \l_tmpa_bool { \int_if_odd_p:n { \l_tmpa_tl } } }
4279 \int_zero_new:N \l_tmpc_int
4280 \int_set:Nn \l_tmpc_int \l_tmpb_tl
4281 \int_do_until:nNnn \l_tmpa_int > \l_tmpc_int
4282 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4283 \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

4284 \bool_if:NT \l_@@_respect_blocks_bool
4285 {
4286 \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
4287 { \@@_intersect_our_row_p:nnnn ###1 }
4288 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4289 }
4290 \tl_set:Nx \l_@@_rows_tl
4291 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
4292 \bool_if:NTF \l_tmpa_bool
4293 {
4294 \tl_if_blank:nF { #3 }
4295 {
4296 \tl_if_empty:nTF { #1 }
4297 \color
4298 { \color [ #1 ] }
4299 { #3 }

```

The command `\@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

4300 \@@_cartesian_path:
4301 \pgfusepath { fill }
4302 }
4303 \bool_set_false:N \l_tmpa_bool
4304 }
4305 {
4306 \tl_if_blank:nF { #4 }
4307 {

```

```

4308         \tl_if_empty:nTF { #1 }
4309         \color
4310         { \color [ #1 ] }
4311         { #4 }

```

The command `\l_@@_cartesian_path:` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_row_tl`.

```

4312         \l_@@_cartesian_path:
4313         \pgfusepath { fill }
4314     }
4315     \bool_set_true:N \l_tmpa_bool
4316 }
4317 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4318 }
4319 }
4320 \endpgfpicture
4321 \group_end:
4322 }

```

```

4323 \cs_new_protected:Npn \l_@@_rowcolors_i:nnnn #1 #2 #3 #4
4324 {
4325     \int_compare:nNnT { #3 } > \l_tmpb_int
4326     { \int_set:Nn \l_tmpb_int { #3 } }
4327 }

```

```

4328 \prg_new_conditional:Nnn \l_@@_not_in_exterior:nnnn p
4329 {
4330     \bool_lazy_or:nnTF
4331     { \int_compare_p:nNn { #4 } = \c_zero_int }
4332     { \int_compare_p:nNn { #2 } = { \l_@@_succ:n { \c_jCol } } }
4333     \prg_return_false:
4334     \prg_return_true:
4335 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4336 \prg_new_conditional:Nnn \l_@@_intersect_our_row:nnnn p
4337 {
4338     \bool_if:nTF
4339     {
4340         \int_compare_p:n { #1 <= \l_tmpa_int }
4341         &&
4342         \int_compare_p:n { \l_tmpa_int <= #3 }
4343     }
4344     \prg_return_true:
4345     \prg_return_false:
4346 }

```

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 `\l_@@_cartesian_path:` which corresponds to a value 0 pt for the radius of the corners. This command is in particular used in `\l_@@_rectanglecolor:nnn` (used in `\l_@@_rectanglecolor`, itself used in `\l_@@_cellcolor`).

```

4347 \cs_new_protected:Npn \l_@@_cartesian_path:n #1
4348 {
4349     \bool_lazy_and:nnT
4350     { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4351     { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4352     {
4353         \l_@@_expand_clist:NN \l_@@_cols_tl \c_jCol
4354         \l_@@_expand_clist:NN \l_@@_rows_tl \c_iRow
4355     }

```


We begin the loop over the columns.

```

4356 \clist_map_inline:Nn \l_@@_cols_tl
4357 {
4358   \tl_set:Nn \l_tmpa_tl { ##1 }
4359   \tl_if_in:NnTF \l_tmpa_tl { - }
4360     { \@@_cut_on_hyphen:w ##1 \q_stop }
4361     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4362   \bool_lazy_or:nnT
4363     { \tl_if_blank_p:V \l_tmpa_tl }
4364     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4365     { \tl_set:Nn \l_tmpa_tl { 1 } }
4366   \bool_lazy_or:nnT
4367     { \tl_if_blank_p:V \l_tmpb_tl }
4368     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4369     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4370   \int_compare:nNnT \l_tmpb_tl > \c@jCol
4371     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_tmpc_tl` will contain the number of column.

```

4372 \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl

```

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.

```

4373 \@@_qpoint:n { col - \l_tmpa_tl }
4374 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4375   { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4376   { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4377 \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4378 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4379 \clist_map_inline:Nn \l_@@_rows_tl
4380 {
4381   \tl_set:Nn \l_tmpa_tl { ####1 }
4382   \tl_if_in:NnTF \l_tmpa_tl { - }
4383     { \@@_cut_on_hyphen:w ####1 \q_stop }
4384     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
4385   \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4386   \tl_if_empty:NT \l_tmpb_tl
4387     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4388   \int_compare:nNnT \l_tmpb_tl > \c@iRow
4389     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

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

```

4390 \seq_if_in:NxF \l_@@_corners_cells_seq
4391 { \l_tmpa_tl - \l_tmpc_tl }
4392 {
4393   \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4394   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4395   \@@_qpoint:n { row - \l_tmpa_tl }
4396   \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4397   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4398   \pgfpathrectanglecorners
4399     { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4400     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4401 }
4402 }
4403 }
4404 }

```

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`).

```

4405 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

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.

```

4406 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4407 {
4408   \clist_set_eq:NN \l_tmpa_clist #1
4409   \clist_clear:N #1
4410   \clist_map_inline:Nn \l_tmpa_clist
4411   {
4412     \tl_set:Nn \l_tmpa_tl { ##1 }
4413     \tl_if_in:NnTF \l_tmpa_tl { - }
4414     { \@@_cut_on_hyphen:w ##1 \q_stop }
4415     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4416     \bool_lazy_or:nnT
4417     { \tl_if_blank_p:V \l_tmpa_tl }
4418     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4419     { \tl_set:Nn \l_tmpa_tl { 1 } }
4420     \bool_lazy_or:nnT
4421     { \tl_if_blank_p:V \l_tmpb_tl }
4422     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4423     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4424     \int_compare:nNnT \l_tmpb_tl > #2
4425     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4426     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4427     { \clist_put_right:Nn #1 { ####1 } }
4428   }
4429 }

```

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

```

4430 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4431 {
4432   \peek_remove_spaces:n
4433   {
4434     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4435     {

```

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`).

```

4436       \cellcolor [ #1 ] { \exp_not:n { #2 } }
4437       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4438     }
4439   }
4440 }

```

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

```

4441 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4442 {
4443   \peek_remove_spaces:n
4444   {
4445     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4446     {
4447       \exp_not:N \rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4448       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4449       { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4450     }
4451   }
4452 }

```

```

4453 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4454 {

```

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).

```
4455 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4456 {
```

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).

```
4457 \tl_gput_left:Nx \g_nicematrix_code_before_tl
4458 {
4459   \exp_not:N \columncolor [ #1 ]
4460   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4461 }
4462 }
4463 }
```

The vertical rules

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
4464 \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`.

```
4465 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4466 {
4467   \int_compare:nNnTF \l_@@_first_col_int = 0
4468   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4469   {
4470     \int_compare:nNnTF \c@jCol = 0
4471     {
4472       \int_compare:nNnF \c@iRow = { -1 }
4473       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4474     }
4475     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4476   }
4477 }
```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
4478 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4479 {
4480   \int_compare:nNnF \c@iRow = 0
4481   { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4482 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the column `#1` (that is to say on the left side). `#2` is the number of consecutive occurrences of `l`.

```
4483 \cs_new_protected:Npn \@@_vline:nn #1 #2
4484 {
```

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

```

4485 \int_compare:nNnT { #1 } < { \c@jCol + 2 }
4486 {
4487   \pgfpicture
4488   \@@_vline_i:nn { #1 } { #2 }
4489   \endpgfpicture
4490 }
4491 }
4492 \cs_new_protected:Npn \@@_vline_i:nn #1 #2
4493 {

```

`\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`.

```

4494 \tl_set:Nx \l_tmpb_tl { #1 }
4495 \tl_clear_new:N \l_tmpc_tl
4496 \int_step_variable:nNn \c@iRow \l_tmpa_tl
4497 {

```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small vertical rule won't be drawn.

```

4498 \bool_gset_true:N \g_tmpa_bool
4499 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4500 { \@@_test_vline_in_block:nnnn ##1 }
4501 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4502 { \@@_test_vline_in_block:nnnn ##1 }
4503 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4504 { \@@_test_vline_in_stroken_block:nnnn ##1 }
4505 \clist_if_empty:NF \l_@@_corners_clist
4506 \@@_test_in_corner_v:
4507 \bool_if:NTF \g_tmpa_bool
4508 {
4509   \tl_if_empty:NT \l_tmpc_tl

```

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

```

4510 { \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl }
4511 }
4512 {
4513   \tl_if_empty:NF \l_tmpc_tl
4514   {
4515     \@@_vline_ii:nnnn
4516     { #1 }
4517     { #2 }
4518     \l_tmpc_tl
4519     { \int_eval:n { \l_tmpa_tl - 1 } }
4520     \tl_clear:N \l_tmpc_tl
4521   }
4522 }
4523 }
4524 \tl_if_empty:NF \l_tmpc_tl
4525 {
4526   \@@_vline_ii:nnnn
4527   { #1 }
4528   { #2 }
4529   \l_tmpc_tl
4530   { \int_use:N \c@iRow }
4531   \tl_clear:N \l_tmpc_tl
4532 }
4533 }

```

```

4534 \cs_new_protected:Npn \@@_test_in_corner_v:

```

```

4535 {
4536   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4537   {
4538     \seq_if_in:NxT
4539     \l_@@_corners_cells_seq
4540     { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4541     { \bool_set_false:N \g_tmpa_bool }
4542   }
4543   {
4544     \seq_if_in:NxT
4545     \l_@@_corners_cells_seq
4546     { \l_tmpa_tl - \l_tmpb_tl }
4547     {
4548       \int_compare:nNnTF \l_tmpb_tl = 1
4549       { \bool_set_false:N \g_tmpa_bool }
4550       {
4551         \seq_if_in:NxT
4552         \l_@@_corners_cells_seq
4553         { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4554         { \bool_set_false:N \g_tmpa_bool }
4555       }
4556     }
4557   }
4558 }

```

#1 is the number of the column; #2 is the number of vertical rules to draw (with potentially a color between); #3 and #4 are the numbers of the rows between which the rule has to be drawn.

```

4559 \cs_new_protected:Npn \@@_vline_ii:nnnn #1 #2 #3 #4
4560 {
4561   \pgfrememberpicturepositiononpagetrue
4562   \pgf@relevantforpicturesizefalse
4563   \@@_qpoint:n { row - #3 }
4564   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4565   \@@_qpoint:n { col - #1 }
4566   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4567   \@@_qpoint:n { row - \@@_succ:n { #4 } }
4568   \dim_set_eq:NN \l_tmpc_dim \pgf@y
4569   \bool_lazy_all:nT
4570   {
4571     { \int_compare_p:nNn { #2 } > 1 }
4572     { \cs_if_exist_p:N \CT@drsc@ } % condition added in version 5.18a
4573     { ! \tl_if_blank_p:V \CT@drsc@ }
4574   }
4575   {
4576     \group_begin:
4577     \CT@drsc@
4578     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4579     \dim_sub:Nn \l_tmpc_dim { 0.5 \arrayrulewidth }
4580     \dim_set:Nn \l_tmpd_dim
4581     { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4582     \pgfpathrectanglecorners
4583     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4584     { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
4585     \pgfusepath { fill }
4586     \group_end:
4587   }
4588   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4589   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4590   \prg_replicate:nn { #2 - 1 }
4591   {
4592     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4593     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4594     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }

```

```

4595     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
4596   }
4597   \CT@arc@
4598   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4599   \pgfsetrectcap
4600   \pgfusepathqstroke
4601 }

```

The following command draws a complete vertical rule in the column #1 (#2 is the number of consecutive rules specified by the number of | in the preamble). This command will be used if there is no block in the array (and the key `corners` is not used).

```

4602 \cs_new_protected:Npn \@@_vline_i_complete:nn #1 #2
4603 { \@@_vline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@iRow } }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

4604 \cs_new_protected:Npn \@@_draw_vlines:
4605 {
4606   \int_step_inline:nnn
4607   {
4608     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4609     1 2
4610   }
4611   {
4612     \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4613     { \@@_succ:n \c@jCol }
4614     \c@jCol
4615   }
4616   {
4617     \tl_if_eq:NnF \l_@@_vlines_clist { all }
4618     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4619     { \@@_vline:nn { ##1 } 1 }
4620   }
4621 }

```

The horizontal rules

The following command will be executed in the `internal-code-after`. The rule will be drawn *before* the row #1. #2 is the number of consecutive occurrences of `\Hline`.

```

4622 \cs_new_protected:Npn \@@_hline:nn #1 #2
4623 {
4624   \pgfpicture
4625   \@@_hline_i:nn { #1 } { #2 }
4626   \endpgfpicture
4627 }
4628 \cs_new_protected:Npn \@@_hline_i:nn #1 #2
4629 {

```

`\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`.

```

4630   \tl_set:Nn \l_tmpa_tl { #1 }
4631   \tl_clear_new:N \l_tmpc_tl
4632   \int_step_variable:nNn \c@jCol \l_tmpb_tl
4633   {

```

The boolean `\g_tmpa_bool` indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```

4634     \bool_gset_true:N \g_tmpa_bool

```

```

4635 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4636 { \@@_test_hline_in_block:nnnn ##1 }
4637 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4638 { \@@_test_hline_in_block:nnnn ##1 }
4639 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4640 { \@@_test_hline_in_stroken_block:nnnn ##1 }
4641 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4642 \bool_if:NTF \g_tmpa_bool
4643 {
4644   \tl_if_empty:NT \l_tmpc_tl

```

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

```

4645   { \tl_set_eq:NN \l_tmpc_tl \l_tmpb_tl }
4646 }
4647 {
4648   \tl_if_empty:NF \l_tmpc_tl
4649   {
4650     \@@_hline_ii:nnnn
4651     { #1 }
4652     { #2 }
4653     \l_tmpc_tl
4654     { \int_eval:n { \l_tmpb_tl - 1 } }
4655     \tl_clear:N \l_tmpc_tl
4656   }
4657 }
4658 }
4659 \tl_if_empty:NF \l_tmpc_tl
4660 {
4661   \@@_hline_ii:nnnn
4662   { #1 }
4663   { #2 }
4664   \l_tmpc_tl
4665   { \int_use:N \c@jCol }
4666   \tl_clear:N \l_tmpc_tl
4667 }
4668 }

4669 \cs_new_protected:Npn \@@_test_in_corner_h:
4670 {
4671   \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
4672   {
4673     \seq_if_in:NxT
4674     \l_@@_corners_cells_seq
4675     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4676     { \bool_set_false:N \g_tmpa_bool }
4677   }
4678   {
4679     \seq_if_in:NxT
4680     \l_@@_corners_cells_seq
4681     { \l_tmpa_tl - \l_tmpb_tl }
4682     {
4683       \int_compare:nNnTF \l_tmpa_tl = 1
4684       { \bool_set_false:N \g_tmpa_bool }
4685       {
4686         \seq_if_in:NxT
4687         \l_@@_corners_cells_seq
4688         { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
4689         { \bool_set_false:N \g_tmpa_bool }
4690       }
4691     }
4692   }
4693 }

```

#1 is the number of the row; #2 is the number of horizontal rules to draw (with potentially a color between); #3 and #4 are the number of the columns between which the rule has to be drawn.

```

4694 \cs_new_protected:Npn \@@_hline_ii:nnnn #1 #2 #3 #4
4695 {
4696   \pgfrememberpicturepositiononpagetrue
4697   \pgf@relevantforpicturesizefalse
4698   \@@_qpoint:n { col - #3 }
4699   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4700   \@@_qpoint:n { row - #1 }
4701   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4702   \@@_qpoint:n { col - \@@_succ:n { #4 } }
4703   \dim_set_eq:NN \l_tmpc_dim \pgf@x
4704   \bool_lazy_and:nnT
4705     { \int_compare_p:nNn { #2 } > 1 }
4706     { ! \tl_if_blank_p:V \CT@drsc@ }
4707     {
4708       \group_begin:
4709       \CT@drsc@
4710       \dim_set:Nn \l_tmpd_dim
4711         { \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth ) * ( #2 - 1 ) }
4712       \pgfpathrectanglecorners
4713         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4714         { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4715       \pgfusepathqfill
4716       \group_end:
4717     }
4718   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4719   \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4720   \prg_replicate:nn { #2 - 1 }
4721   {
4722     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4723     \dim_sub:Nn \l_tmpb_dim \doublerulesep
4724     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4725     \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
4726   }
4727   \CT@arc@
4728   \pgfsetlinewidth { 1.1 \arrayrulewidth }
4729   \pgfsetrectcap
4730   \pgfusepathqstroke
4731 }

4732 \cs_new_protected:Npn \@@_hline_i_complete:nn #1 #2
4733 { \@@_hline_ii:nnnn { #1 } { #2 } 1 { \int_use:N \c@jCol } }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual drawn determined by commands such as `\Cdots` and in the corners (if the key `corners` is used).

```

4734 \cs_new_protected:Npn \@@_draw_hlines:
4735 {
4736   \int_step_inline:nnn
4737     {
4738       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4739       1 2
4740     }
4741     {
4742       \bool_if:nTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4743       { \@@_succ:n \c@iRow }
4744       \c@iRow
4745     }
4746     {
4747       \tl_if_eq:NnF \l_@@_hlines_clist { all }
4748       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }

```



```

4749         { \@@_hline:nn { ##1 } 1 }
4750     }
4751 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

4752 \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.

```

4753 \cs_set:Npn \@@_Hline_i:n #1
4754 {
4755     \peek_meaning_ignore_spaces:NTF \Hline
4756     { \@@_Hline_ii:nn { #1 + 1 } }
4757     { \@@_Hline_iii:n { #1 } }
4758 }
4759 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
4760 \cs_set:Npn \@@_Hline_iii:n #1
4761 {
4762     \skip_vertical:n
4763     {
4764         \arrayrulewidth * ( #1 )
4765         + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
4766     }
4767     \tl_gput_right:Nx \g_@@_internal_code_after_tl
4768     { \@@_hline:nn { \@@_succ:n { \c@iRow } } { #1 } }
4769     \ifnum 0 = ` { \fi }
4770 }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments `#1`, `#2`, `#3` and `#4`. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to `false`.

```

4771 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4
4772 {
4773     \bool_lazy_all:nT
4774     {
4775         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
4776         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4777         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4778         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4779     }
4780     { \bool_gset_false:N \g_tmpa_bool }
4781 }

```

The same for vertical rules.

```

4782 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4
4783 {
4784     \bool_lazy_all:nT
4785     {
4786         { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4787         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4788         { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
4789         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4790     }
4791     { \bool_gset_false:N \g_tmpa_bool }
4792 }
4793 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
4794 {
4795     \bool_lazy_all:nT
4796     {

```

```

4797     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4798     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
4799     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4800     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
4801   }
4802   { \bool_gset_false:N \g_tmpa_bool }
4803 }
4804 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
4805 {
4806   \bool_lazy_all:nT
4807   {
4808     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
4809     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
4810     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
4811     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
4812   }
4813   { \bool_gset_false:N \g_tmpa_bool }
4814 }

```

The key corners

When the key `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.

```

4815 \cs_new_protected:Npn \@@_compute_corners:
4816 {

```

The sequence `\l_@@_corners_cells_seq` will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```

4817   \seq_clear_new:N \l_@@_corners_cells_seq
4818   \clist_map_inline:Nn \l_@@_corners_clist
4819   {
4820     \str_case:nnF { ##1 }
4821     {
4822       { NW }
4823       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
4824       { NE }
4825       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
4826       { SW }
4827       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
4828       { SE }
4829       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
4830     }
4831     { \@@_error:nn { bad~corner } { ##1 } }
4832   }

```

Even if the user has used the key `corners` (or the key `hvlines-except-corners`), the list of cells in the corners may be empty.

```

4833   \seq_if_empty:NF \l_@@_corners_cells_seq
4834   {

```

You write on the aux file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

4835     \tl_gput_right:Nx \g_@@_aux_tl
4836     {
4837       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
4838       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
4839     }
4840   }
4841 }

```

“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 `\@@_compute_a_corner:nnnnnn` are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```
4842 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
4843 {
```

For the explanations and the name of the variables, we consider that we are computing the left-upper corner.

First, we try to determine which is the last empty cell (and not in a block: we won’t add that precision any longer) in the column of number 1. The flag `\l_tmpa_bool` will be raised when a non-empty cell is found.

```
4844   \bool_set_false:N \l_tmpa_bool
4845   \int_zero_new:N \l_@@_last_empty_row_int
4846   \int_set:Nn \l_@@_last_empty_row_int { #1 }
4847   \int_step_inline:nnnn { #1 } { #3 } { #5 }
4848   {
4849     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
4850     \bool_lazy_or:nnTF
4851     {
4852       \cs_if_exist_p:c
4853       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
4854     }
4855     \l_tmpb_bool
4856     { \bool_set_true:N \l_tmpa_bool }
4857     {
4858       \bool_if:NF \l_tmpa_bool
4859       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
4860     }
4861   }
```

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

```
4862   \bool_set_false:N \l_tmpa_bool
4863   \int_zero_new:N \l_@@_last_empty_column_int
4864   \int_set:Nn \l_@@_last_empty_column_int { #2 }
4865   \int_step_inline:nnnn { #2 } { #4 } { #6 }
4866   {
4867     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
4868     \bool_lazy_or:nnTF
4869     \l_tmpb_bool
4870     {
4871       \cs_if_exist_p:c
4872       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
4873     }
4874     { \bool_set_true:N \l_tmpa_bool }
4875     {
4876       \bool_if:NF \l_tmpa_bool
4877       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
4878     }
4879   }
```

Now, we loop over the rows.

```
4880   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
4881   {
```

We treat the row number `##1` with another loop.

```

4882     \bool_set_false:N \l_tmpa_bool
4883     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
4884     {
4885         \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
4886         \bool_lazy_or:nnTF
4887             \l_tmpb_bool
4888             {
4889                 \cs_if_exist_p:c
4890                     { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
4891             }
4892         { \bool_set_true:N \l_tmpa_bool }
4893         {
4894             \bool_if:NF \l_tmpa_bool
4895             {
4896                 \int_set:Nn \l_@@_last_empty_column_int { ####1 }
4897                 \seq_put_right:Nn
4898                     \l_@@_corners_cells_seq
4899                     { ##1 - ####1 }
4900             }
4901         }
4902     }
4903 }
4904 }
```

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`).

```

4905 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
4906 {
4907     \int_set:Nn \l_tmpa_int { #1 }
4908     \int_set:Nn \l_tmpb_int { #2 }
4909     \bool_set_false:N \l_tmpb_bool
4910     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4911         { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
4912 }
4913 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6
4914 {
4915     \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
4916     {
4917         \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
4918         {
4919             \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
4920             {
4921                 \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
4922                 { \bool_set_true:N \l_tmpb_bool }
4923             }
4924         }
4925     }
4926 }
```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```

4927 \cs_new:Npn \@@_hdottedline:
4928 {
```

```

4929 \noalign { \skip_vertical:N 2\l_@@_radius_dim }
4930 \@@_hdottedline_i:
4931 }

```

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

```

4932 \cs_new_protected:Npn \@@_hdottedline_i:
4933 {

```

We write in the code-after the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```

4934 \tl_gput_right:Nx \g_@@_internal_code_after_tl
4935 { \@@_hdottedline:n { \int_use:N \c@iRow } }
4936 }

```

The command `\@@_hdottedline:n` is the command written in the `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```

4937 \AtBeginDocument
4938 {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That's why we construct now a version of `\@@_hdottedline:n` with the right environment (`\begin{pgfpicture}\end{pgfpicture}` or `\begin{tikzpicture}...\end{tikzpicture}`).

```

4939 \cs_new_protected:Npx \@@_hdottedline:n #1
4940 {
4941 \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
4942 \bool_set_true:N \exp_not:N \l_@@_final_open_bool
4943 \c_@@_pgfortikzpicture_tl
4944 \@@_hdottedline_i:n { #1 }
4945 \c_@@_endpgfortikzpicture_tl
4946 }
4947 }

```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```

4948 \cs_new_protected:Npn \@@_hdottedline_i:n #1
4949 {
4950 \pgfrememberpicturepositiononpagetrue
4951 \@@_qpoint:n { row - #1 }

```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```

4952 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4953 \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
4954 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

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

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

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

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

4955 \@@_qpoint:n { col - 1 }
4956 \dim_set:Nn \l_@@_x_initial_dim
4957 {
4958   \pgf@x +

```

We do a reduction by `\arraycolsep` for the environments with delimiters (and not for the other).

```

4959   \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4960   - \l_@@_left_margin_dim
4961 }
4962 \@@_qpoint:n { col - \@@_succ:n \c@jCol }
4963 \dim_set:Nn \l_@@_x_final_dim
4964 {
4965   \pgf@x -
4966   \bool_if:NTF \l_@@_NiceArray_bool \c_zero_dim \arraycolsep
4967   + \l_@@_right_margin_dim
4968 }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_inter_dots_dim` is *ad hoc* for a better result.

```

4969 \tl_if_eq:NnF \g_@@_left_delim_tl (
4970   { \dim_gadd:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
4971 \tl_if_eq:NnF \g_@@_right_delim_tl )
4972 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }

```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “.” in the preamble. That’s why we impose the style `standard`.

```

4973 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
4974 \@@_draw_line:
4975 }

```

Vertical dotted lines

```

4976 \cs_new_protected:Npn \@@_vdottedline:n #1
4977 {
4978   \bool_set_true:N \l_@@_initial_open_bool
4979   \bool_set_true:N \l_@@_final_open_bool

```

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

```

4980   \bool_if:NTF \c_@@_tikz_loaded_bool
4981   {
4982     \tikzpicture
4983     \@@_vdottedline_i:n { #1 }
4984     \endtikzpicture
4985   }
4986   {
4987     \pgfpicture
4988     \@@_vdottedline_i:n { #1 }
4989     \endpgfpicture
4990   }
4991 }

```

```

4992 \cs_new_protected:Npn \@@_vdottedline_i:n #1
4993 {

```

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.

```

4994 \CT@arc@
4995 \pgfrememberpicturepositiononpagetrue
4996 \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }

```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

```
4997 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
4998 \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
4999 \@@_qpoint:n { row - 1 }
```

We arbitrary decrease the height of the dotted line by a quantity equal to `\l_@@_inter_dots_dim` in order to improve the visual impact.

```
5000 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
5001 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
5002 \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }
```

Up to now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier “:” in the preamble. That’s why we impose the style `standard`.

```
5003 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
5004 \@@_draw_line:
5005 }
```

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.

```
5006 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
5007 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5008 {
5009   auto-columns-width .code:n =
5010   {
5011     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5012     \dim_gzero_new:N \g_@@_max_cell_width_dim
5013     \bool_set_true:N \l_@@_auto_columns_width_bool
5014   }
5015 }

5016 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
5017 {
5018   \int_gincr:N \g_@@_NiceMatrixBlock_int
5019   \dim_zero:N \l_@@_columns_width_dim
5020   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5021   \bool_if:NT \l_@@_block_auto_columns_width_bool
5022   {
5023     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5024     {
5025       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
5026       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5027     }
5028   }
5029 }
```

At the end of the environment {NiceMatrixBlock}, we write in the main .aux file instructions for the column width of all the environments of the block (that’s why we have stored the number of the first environment of the block in the counter `\l_@@_first_env_block_int`).

```
5030 {
5031   \bool_if:NT \l_@@_block_auto_columns_width_bool
5032   {
5033     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5034     \iow_shipout:Nx \@mainaux
```

```

5035     {
5036         \cs_gset:cpn
5037         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

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

```

5038         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5039     }
5040     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5041 }
5042 }

```

The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

5043 \cs_generate_variant:Nn \dim_min:nn { v n }
5044 \cs_generate_variant:Nn \dim_max:nn { v n }

```

The following command is called in `\@@_use_arraybox_with_notes_c:` just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

5045 \cs_new_protected:Npn \@@_create_extra_nodes:
5046 {
5047     \bool_if:nTF \l_@@_medium_nodes_bool
5048     {
5049         \bool_if:nTF \l_@@_large_nodes_bool
5050         \@@_create_medium_and_large_nodes:
5051         \@@_create_medium_nodes:
5052     }
5053     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5054 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row i , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal y -value of all the cells of the row i . The dimension `l_@@_row_i_max_dim` is the maximal y -value of all the cells of the row i .

Similarly, for each column j , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal x -value of all the cells of the column j . The dimension `l_@@_column_j_max_dim` is the maximal x -value of all the cells of the column j .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

5055 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5056 {
5057     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5058     {
5059         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
5060         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
5061         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
5062         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
5063     }
5064     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5065     {

```



```

5066 \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
5067 \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
5068 \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
5069 \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
5070 }

```

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

```

5071 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5072 {
5073   \int_step_variable:nnNn
5074   \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

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.

```

5075   {
5076     \cs_if_exist:cT
5077     { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

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

```

5078   {
5079     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5080     \dim_set:cn { l_@@_row_\@@_i: _min_dim }
5081     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5082     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5083     {
5084       \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5085       { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5086     }

```

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

```

5087     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5088     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
5089     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
5090     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5091     {
5092       \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
5093       { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
5094     }
5095   }
5096 }
5097 }

```

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

```

5098 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5099 {
5100   \dim_compare:nNnT
5101   { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
5102   {
5103     \@@_qpoint:n { row - \@@_i: - base }
5104     \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
5105     \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
5106   }
5107 }
5108 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5109 {
5110   \dim_compare:nNnT
5111   { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
5112   {
5113     \@@_qpoint:n { col - \@@_j: }
5114     \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@x
5115     \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@x
5116   }

```

```

5117     }
5118 }

```

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

```

5119 \cs_new_protected:Npn \@@_create_medium_nodes:
5120 {
5121   \pgfpicture
5122     \pgfrememberpicturepositiononpagetrue
5123     \pgf@relevantforpicturesizefalse
5124     \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

5125     \tl_set:Nn \l_@@_suffix_tl { -medium }
5126     \@@_create_nodes:
5127   \endpgfpicture
5128 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁶⁴. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

5129 \cs_new_protected:Npn \@@_create_large_nodes:
5130 {
5131   \pgfpicture
5132     \pgfrememberpicturepositiononpagetrue
5133     \pgf@relevantforpicturesizefalse
5134     \@@_computations_for_medium_nodes:
5135     \@@_computations_for_large_nodes:
5136     \tl_set:Nn \l_@@_suffix_tl { -large }
5137     \@@_create_nodes:
5138   \endpgfpicture
5139 }

5140 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5141 {
5142   \pgfpicture
5143     \pgfrememberpicturepositiononpagetrue
5144     \pgf@relevantforpicturesizefalse
5145     \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

5146     \tl_set:Nn \l_@@_suffix_tl { -medium }
5147     \@@_create_nodes:
5148     \@@_computations_for_large_nodes:
5149     \tl_set:Nn \l_@@_suffix_tl { -large }
5150     \@@_create_nodes:
5151   \endpgfpicture
5152 }

```

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.

```

5153 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5154 {
5155   \int_set:Nn \l_@@_first_row_int 1
5156   \int_set:Nn \l_@@_first_col_int 1

```

⁶⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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`.

```

5157 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5158 {
5159   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5160   {
5161     (
5162       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5163       \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5164     )
5165     / 2
5166   }
5167   \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5168   { l_@@_row_\@@_i: _ min_dim }
5169 }
5170 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5171 {
5172   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5173   {
5174     (
5175       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5176       \dim_use:c
5177       { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5178     )
5179     / 2
5180   }
5181   \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5182   { l_@@_column _ \@@_j: _ max _ dim }
5183 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

5184 \dim_sub:cn
5185 { l_@@_column _ 1 _ min _ dim }
5186 \l_@@_left_margin_dim
5187 \dim_add:cn
5188 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5189 \l_@@_right_margin_dim
5190 }

```

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).

```

5191 \cs_new_protected:Npn \@@_create_nodes:
5192 {
5193   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5194   {
5195     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5196     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

5197 \@@_pgf_rect_node:nnnnn
5198 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5199 { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
5200 { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
5201 { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
5202 { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
5203 \str_if_empty:NF \l_@@_name_str
5204 {
5205   \pgfnodealias
5206   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }

```

```

5207         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5208     }
5209 }
5210 }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

5211 \seq_mapthread_function:NNN
5212   \g_@@_multicolumn_cells_seq
5213   \g_@@_multicolumn_sizes_seq
5214   \@@_node_for_multicolumn:nn
5215 }

```

```

5216 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5217 {
5218   \cs_set_nopar:Npn \@@_i: { #1 }
5219   \cs_set_nopar:Npn \@@_j: { #2 }
5220 }

```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

5221 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5222 {
5223   \@@_extract_coords_values: #1 \q_stop
5224   \@@_pgf_rect_node:nnnnn
5225     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5226     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
5227     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
5228     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
5229     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
5230   \str_if_empty:NF \l_@@_name_str
5231   {
5232     \pgfnodealias
5233       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5234       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5235   }
5236 }

```

The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

```

5237 \keys_define:nn { NiceMatrix / Block / FirstPass }
5238 {
5239   l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5240   l .value_forbidden:n = true ,
5241   r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5242   r .value_forbidden:n = true ,
5243   c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5244   c .value_forbidden:n = true ,
5245   L .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5246   L .value_forbidden:n = true ,
5247   R .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5248   R .value_forbidden:n = true ,
5249   C .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,

```

```

5250 C .value_forbidden:n = true ,
5251 t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5252 t .value_forbidden:n = true ,
5253 b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5254 b .value_forbidden:n = true ,
5255 color .tl_set:N = \l_@@_color_tl ,
5256 color .value_required:n = true ,
5257 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>` (for TeX instructions put before the math mode of the label and before the beginning of the small array of the block). It's mandatory to use an expandable command.

```

5258 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } m }
5259 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

5260 \peek_remove_spaces:n
5261 {
5262   \tl_if_blank:nTF { #2 }
5263   { \@@_Block_i 1-1 \q_stop }
5264   { \@@_Block_i #2 \q_stop }
5265   { #1 } { #3 } { #4 }
5266 }
5267 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5268 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

Now, the arguments have been extracted: `#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of key-values, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```

5269 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5270 {

```

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).

```

5271 \bool_lazy_or:nnTF
5272 { \tl_if_blank_p:n { #1 } }
5273 { \str_if_eq_p:nn { #1 } { * } }
5274 { \int_set:Nn \l_tmpa_int { 100 } }
5275 { \int_set:Nn \l_tmpa_int { #1 } }
5276 \bool_lazy_or:nnTF
5277 { \tl_if_blank_p:n { #2 } }
5278 { \str_if_eq_p:nn { #2 } { * } }
5279 { \int_set:Nn \l_tmpb_int { 100 } }
5280 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5281 \int_compare:nNnTF \l_tmpb_int = 1
5282 {
5283   \tl_if_empty:NTF \l_@@_cell_type_tl
5284   { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5285   { \tl_set_eq:NN \l_@@_hpos_of_block_tl \l_@@_cell_type_tl }
5286 }
5287 { \tl_set:Nn \l_@@_hpos_of_block_tl c }

```

The value of `\l_@@_hpos_of_block_tl` may be modified by the keys of the command `\Block` that we will analyze now.

```

5288 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5289 \tl_set:Nx \l_tmpa_tl
5290 {
5291   { \int_use:N \c@iRow }
5292   { \int_use:N \c@jCol }
5293   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5294   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5295 }

```

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).

```

5296 \bool_if:nTF
5297 {
5298   (
5299     \int_compare_p:nNn { \l_tmpa_int } = 1
5300     ||
5301     \int_compare_p:nNn { \l_tmpb_int } = 1
5302   )
5303   && ! \tl_if_empty_p:n { #5 }
5304 }
5305 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5306 { \exp_args:Nxx \@@_Block_v:nnnnn }
5307 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5308 }

```

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).

```

5309 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5310 {
5311   \int_gincr:N \g_@@_block_box_int
5312   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5313   {
5314     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5315     {
5316       \@@_actually_diagbox:nnnnnn
5317       { \int_use:N \c@iRow }
5318       { \int_use:N \c@jCol }
5319       { \int_eval:n { \c@iRow + #1 - 1 } }
5320       { \int_eval:n { \c@jCol + #2 - 1 } }
5321       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5322     }
5323   }
5324   \box_gclear_new:c
5325   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
5326   \hbox_gset:cn
5327   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
5328   {

```

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}`).

```

5329 \tl_if_empty:NTF \l_@@_color_tl
5330 { \int_compare:nNnT { #2 } = 1 \set@color }
5331 { \color { \l_@@_color_tl } }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even 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`.

```

5332 \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5333 \group_begin:
5334 \cs_set:Npn \arraystretch { 1 }
5335 \dim_set_eq:NN \extrarowheight \c_zero_dim
5336 #4

```

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).

```

5337 \bool_if:NT \g_@@_rotate_bool { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5338 \bool_if:NTF \l_@@_NiceTabular_bool
5339 {
5340   \use:x
5341   {
5342     \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5343     { @ { } \l_@@_hpos_of_block_tl @ { } }
5344   }
5345   #5
5346   \end { tabular }
5347 }
5348 {
5349   \c_math_toggle_token
5350   \use:x
5351   {
5352     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5353     { @ { } \l_@@_hpos_of_block_tl @ { } }
5354   }
5355   #5
5356   \end { array }
5357   \c_math_toggle_token
5358 }
5359 \group_end:
5360 }
5361 \bool_if:NT \g_@@_rotate_bool
5362 {
5363   \box_grotate:cn
5364   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5365   { 90 }
5366   \bool_gset_false:N \g_@@_rotate_bool
5367 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5368 \int_compare:nNnT { #2 } = 1
5369 {
5370   \dim_gset:Nn \g_@@_blocks_wd_dim
5371   {
5372     \dim_max:nn
5373     \g_@@_blocks_wd_dim
5374     {
5375       \box_wd:c
5376       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5377     }
5378   }
5379 }

```

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.

```

5380 \int_compare:nNnT { #1 } = 1
5381 {
5382   \dim_gset:Nn \g_@@_blocks_ht_dim
5383   {
5384     \dim_max:nn
5385     \g_@@_blocks_ht_dim
5386     {
5387       \box_ht:c
5388       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5389     }
5390   }
5391   \dim_gset:Nn \g_@@_blocks_dp_dim
5392   {
5393     \dim_max:nn
5394     \g_@@_blocks_dp_dim
5395     {
5396       \box_dp:c
5397       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5398     }
5399   }
5400 }
5401 \seq_gput_right:Nx \g_@@_blocks_seq
5402 {
5403   \l_tmpa_tl

```

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_of_block_tl. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of \l_@@_hpos_of_block_tl, which is fixed by the type of current column.

```

5404   { \exp_not:n { #3 } , \l_@@_hpos_of_block_tl }
5405   {
5406     \box_use_drop:c
5407     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5408   }
5409 }
5410 }

```

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.

```

5411 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5412 {
5413   \seq_gput_right:Nx \g_@@_blocks_seq
5414   {
5415     \l_tmpa_tl
5416     { \exp_not:n { #3 } }
5417     \exp_not:n
5418     {
5419       {
5420         \bool_if:NTF \l_@@_NiceTabular_bool
5421         {
5422           \group_begin:
5423           \cs_set:Npn \arraystretch { 1 }
5424           \dim_set_eq:NN \extrarowheight \c_zero_dim
5425           #4

```

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).


```

5426 \bool_if:NT \g_@@_rotate_bool
5427 { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5428 \use:x
5429 {
5430 \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5431 { @ { } \l_@@_hpos_of_block_tl @ { } }
5432 }
5433 #5
5434 \end { tabular }
5435 \group_end:
5436 }
5437 {
5438 \group_begin:
5439 \cs_set:Npn \arraystretch { 1 }
5440 \dim_set_eq:NN \extrarowheight \c_zero_dim
5441 #4
5442 \bool_if:NT \g_@@_rotate_bool
5443 { \tl_set:Nn \l_@@_hpos_of_block_tl c }
5444 \c_math_toggle_token
5445 \use:x
5446 {
5447 \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5448 { @ { } \l_@@_hpos_of_block_tl @ { } }
5449 }
5450 #5
5451 \end { array }
5452 \c_math_toggle_token
5453 \group_end:
5454 }
5455 }
5456 }
5457 }
5458 }

```

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).

```

5459 \keys_define:nn { NiceMatrix / Block / SecondPass }
5460 {
5461 tikz .code:n =
5462 \bool_if:NTF \c_@@_tikz_loaded_bool
5463 { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5464 { \@@_error:n { tikz-key-without~tikz } } ,
5465 tikz .value_required:n = true ,
5466 fill .tl_set:N = \l_@@_fill_tl ,
5467 fill .value_required:n = true ,
5468 draw .tl_set:N = \l_@@_draw_tl ,
5469 draw .default:n = default ,
5470 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5471 rounded-corners .default:n = 4 pt ,
5472 color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5473 color .value_required:n = true ,
5474 borders .clist_set:N = \l_@@_borders_clist ,
5475 borders .value_required:n = true ,
5476 hvlines .bool_set:N = \l_@@_hvlines_block_bool ,
5477 hvlines .default:n = true ,
5478 line-width .dim_set:N = \l_@@_line_width_dim ,
5479 line-width .value_required:n = true ,
5480 l .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l ,
5481 l .value_forbidden:n = true ,
5482 r .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r ,
5483 r .value_forbidden:n = true ,

```

```

5484 c .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c ,
5485 c .value_forbidden:n = true ,
5486 L .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl l
5487       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5488 L .value_forbidden:n = true ,
5489 R .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl r
5490       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5491 R .value_forbidden:n = true ,
5492 C .code:n = \tl_set:Nn \l_@@_hpos_of_block_tl c
5493       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5494 C .value_forbidden:n = true ,
5495 t .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl t ,
5496 t .value_forbidden:n = true ,
5497 b .code:n = \tl_set:Nn \l_@@_vpos_of_block_tl b ,
5498 b .value_forbidden:n = true ,
5499 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
5500 }

```

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.

```

5501 \cs_new_protected:Npn \@@_draw_blocks:
5502 {
5503   \cs_set_eq:NN \ialign \@@_old_ialign:
5504   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
5505 }
5506 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5507 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5508   \int_zero_new:N \l_@@_last_row_int
5509   \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format *i-j*. However, the user is allowed to omit *i* or *j* (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

5510   \int_compare:nNnTF { #3 } > { 99 }
5511     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
5512     { \int_set:Nn \l_@@_last_row_int { #3 } }
5513   \int_compare:nNnTF { #4 } > { 99 }
5514     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
5515     { \int_set:Nn \l_@@_last_col_int { #4 } }
5516   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
5517     {
5518       \int_compare:nTF
5519         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
5520         {
5521           \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
5522           \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
5523           \group_begin:
5524           \globaldefs = 1
5525           \@@_msg_redirect_name:nn { columns~not~used } { none }
5526           \group_end:
5527         }
5528         { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
5529     }
5530   {
5531     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int

```

```

5532         { \msg_error:nnnn { nicematrix } { Block-too~large~1 } { #1 } { #2 } }
5533         { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
5534     }
5535 }

5536 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
5537 {

```

The sequence of the positions of the blocks will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

5538     \seq_gput_left:Nn \g_@@_pos_of_blocks_seq { { #1 } { #2 } { #3 } { #4 } }

```

The group is for the keys.

```

5539     \group_begin:
5540     \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

5541     \bool_if:NT \l_@@_hvlines_block_bool
5542     {
5543         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5544         {
5545             \@@_hvlines_block:nnn
5546             { \exp_not:n { #5 } }
5547             { #1 - #2 }
5548             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5549         }
5550     }

5551     \tl_if_empty:NF \l_@@_draw_tl
5552     {
5553         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5554         {
5555             \@@_stroke_block:nnn
5556             { \exp_not:n { #5 } }
5557             { #1 - #2 }
5558             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5559         }
5560         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
5561         { { #1 } { #2 } { #3 } { #4 } }
5562     }

5563     \clist_if_empty:NF \l_@@_borders_clist
5564     {
5565         \tl_gput_right:Nx \g_nicematrix_code_after_tl
5566         {
5567             \@@_stroke_borders_block:nnn
5568             { \exp_not:n { #5 } }
5569             { #1 - #2 }
5570             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5571         }
5572     }

5573     \tl_if_empty:NF \l_@@_fill_tl
5574     {

```

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`.

```

5575         \exp_last_unbraced:NV \@@_extract_brackets \l_@@_fill_tl \q_stop
5576         \tl_gput_right:Nx \g_nicematrix_code_before_tl
5577         {
5578             \exp_not:N \roundedrectanglecolor
5579             [ \l_tmpa_tl ]
5580             { \exp_not:V \l_tmpb_tl }
5581             { #1 - #2 }
5582             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
5583             { \dim_use:N \l_@@_rounded_corners_dim }
5584         }
5585     }

```

```

5586 \seq_if_empty:NF \l_@@_tikz_seq
5587 {
5588   \tl_gput_right:Nx \g_nicematrix_code_before_tl
5589   {
5590     \@@_block_tikz:nnnnn
5591     { #1 }
5592     { #2 }
5593     { \int_use:N \l_@@_last_row_int }
5594     { \int_use:N \l_@@_last_col_int }
5595     { \seq_use:Nn \l_@@_tikz_seq { , } }
5596   }
5597 }

5598 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5599 {
5600   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5601   {
5602     \@@_actually_diagbox:nnnnnn
5603     { #1 }
5604     { #2 }
5605     { \int_use:N \l_@@_last_row_int }
5606     { \int_use:N \l_@@_last_col_int }
5607     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
5608   }
5609 }

5610 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
5611 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one & \\
                        &      & two & \\
three                  & four & five & \\
six                    & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

5612 \pgfpicture
5613   \pgfrememberpicturepositiononpagetrue
5614   \pgf@relevantforpicturesizefalse
5615   \@@_qpoint:n { row - #1 }
5616   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5617   \@@_qpoint:n { col - #2 }
5618   \dim_set_eq:NN \l_tmpb_dim \pgf@x
5619   \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
5620   \dim_set_eq:NN \l_tmpc_dim \pgf@y
5621   \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5622   \dim_set_eq:NN \l_tmpd_dim \pgf@x

```

We construct the node for the block with the name (#1-#2-block).

The function `\@@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.

```

5623 \begin { pgfscope }
5624 \@@pgf_rect_node:nnnnn
5625 { \@@_env: - #1 - #2 - block }
5626 \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
5627 \end { pgfscope }

```

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 the keys L, C or R is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don’t need to create that node since the normal node is used to put the label.

```

5628 \bool_if:NF \l_@@_hpos_of_block_cap_bool
5629 {
5630 \dim_set_eq:NN \l_tmpb_dim \c_max_dim

```

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.

```

5631 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5632 {

```

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.

```

5633 \cs_if_exist:cT
5634 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
5635 {
5636 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5637 {
5638 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
5639 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
5640 }
5641 }
5642 }

```

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

```

5643 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
5644 {
5645 \@@_qpoint:n { col - #2 }
5646 \dim_set_eq:NN \l_tmpb_dim \pgf@x
5647 }
5648 \dim_set:Nn \l_tmpd_dim { - \c_max_dim }
5649 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
5650 {
5651 \cs_if_exist:cT
5652 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5653 {
5654 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
5655 {
5656 \pgfpointanchor
5657 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
5658 { east }
5659 \dim_set:Nn \l_tmpd_dim { \dim_max:nn \l_tmpd_dim \pgf@x }
5660 }
5661 }
5662 }
5663 \dim_compare:nNnT \l_tmpd_dim = { - \c_max_dim }
5664 {
5665 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
5666 \dim_set_eq:NN \l_tmpd_dim \pgf@x
5667 }
5668 \@@pgf_rect_node:nnnnn

```

```

5669         { \@@_env: - #1 - #2 - block - short }
5670         \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpe_dim
5671     }

```

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.

```

5672     \bool_if:NT \l_@@_medium_nodes_bool
5673     {
5674         \@@_pgf_rect_node:nnn
5675         { \@@_env: - #1 - #2 - block - medium }
5676         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
5677         {
5678             \pgfpointanchor
5679             { \@@_env:
5680               - \int_use:N \l_@@_last_row_int
5681               - \int_use:N \l_@@_last_col_int - medium
5682             }
5683             { south-east }
5684         }
5685     }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

5686     \int_compare:nNnTF { #1 } = { #3 }
5687     {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

5688         \int_compare:nNnTF { #1 } = 0
5689         { \l_@@_code_for_first_row_tl }
5690         {
5691             \int_compare:nNnT { #1 } = \l_@@_last_row_int
5692             \l_@@_code_for_last_row_tl
5693         }

```

If the block has only one row, we want the label of the block perfectly aligned on the baseline of the row. That’s why we have constructed a `\pgfcoordinate` on the baseline of the row, in the first column of the array. Now, we retrieve the y -value of that node and we store it in `\l_tmpa_dim`.

```

5694         \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

5695         \pgfpointanchor
5696         {
5697             \@@_env: - #1 - #2 - block
5698             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5699         }
5700         {
5701             \str_case:Vn \l_@@_hpos_of_block_tl
5702             {
5703                 c { center }
5704                 l { west }
5705                 r { east }
5706             }
5707         }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

5708         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
5709         \pgfset { inner-sep = \c_zero_dim }
5710         \pgfnode
5711         { rectangle }
5712         {
5713             \str_case:Vn \l_@@_hpos_of_block_tl
5714             {
5715                 c { base }
5716                 l { base-west }
5717                 r { base-east }
5718             }

```

```

5719     }
5720     { \box_use_drop:N \l_@@_cell_box } { } { }
5721 }

```

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`).

```

5722 {
If we are in the first column, we must put the block as if it was with the key r.
5723     \int_compare:nNnT { #2 } = 0
5724     { \tl_set:Nn \l_@@_hpos_of_block_tl r }
5725     \bool_if:nT \g_@@_last_col_found_bool
5726     {
5727         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5728         { \tl_set:Nn \l_@@_hpos_of_block_tl l }
5729     }
5730     \pgftransformshift
5731     {
5732         \pgfpointanchor
5733         {
5734             \@@_env: - #1 - #2 - block
5735             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
5736         }
5737         {
5738             \str_case:Vn \l_@@_hpos_of_block_tl
5739             {
5740                 c { center }
5741                 l { west }
5742                 r { east }
5743             }
5744         }
5745     }
5746     \pgfset { inner-sep = \c_zero_dim }
5747     \pgfnode
5748     { rectangle }
5749     {
5750         \str_case:Vn \l_@@_hpos_of_block_tl
5751         {
5752             c { center }
5753             l { west }
5754             r { east }
5755         }
5756     }
5757     { \box_use_drop:N \l_@@_cell_box } { } { }
5758 }
5759 \endpgfpicture
5760 \group_end:
5761 }

```

```

5762 \NewDocumentCommand \@@_extract_brackets { 0 { } }
5763 {
5764     \tl_set:Nn \l_tmpa_tl { #1 }
5765     \@@_store_in_tmpb_tl
5766 }
5767 \cs_new_protected:Npn \@@_store_in_tmpb_tl #1 \q_stop
5768 { \tl_set:Nn \l_tmpb_tl { #1 } }

```

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).

```

5769 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
5770 {

```

```

5771 \group_begin:
5772 \tl_clear:N \l_@@_draw_tl
5773 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5774 \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
5775 \pgfpicture
5776 \pgfrememberpicturepositiononpagetrue
5777 \pgf@relevantforpicturesizefalse
5778 \tl_if_empty:NF \l_@@_draw_tl
5779 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

5780     \str_if_eq:VnTF \l_@@_draw_tl { default }
5781     { \CT@arc@ }
5782     { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
5783   }
5784   \pgfsetcornersarced
5785   {
5786     \pgfpoint
5787     { \dim_use:N \l_@@_rounded_corners_dim }
5788     { \dim_use:N \l_@@_rounded_corners_dim }
5789   }
5790   \@@_cut_on_hyphen:w #2 \q_stop
5791   \bool_lazy_and:nnT
5792   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
5793   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
5794   {
5795     \@@_qpoint:n { row - \l_tmpa_tl }
5796     \dim_set:Nn \l_tmpb_dim { \pgf@y }
5797     \@@_qpoint:n { col - \l_tmpb_tl }
5798     \dim_set:Nn \l_tmpc_dim { \pgf@x }
5799     \@@_cut_on_hyphen:w #3 \q_stop
5800     \int_compare:nNnT \l_tmpa_tl > \c@iRow
5801     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
5802     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5803     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5804     \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
5805     \dim_set:Nn \l_tmpa_dim { \pgf@y }
5806     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
5807     \dim_set:Nn \l_tmpd_dim { \pgf@x }
5808     \pgfpathrectanglecorners
5809     { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
5810     { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
5811     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

5812     \pgfusepath { stroke }
5813   }
5814   \endpgfpicture
5815   \group_end:
5816 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

5817 \keys_define:nn { NiceMatrix / BlockStroke }
5818 {
5819   color .tl_set:N = \l_@@_draw_tl ,
5820   draw .tl_set:N = \l_@@_draw_tl ,
5821   draw .default:n = default ,
5822   line-width .dim_set:N = \l_@@_line_width_dim ,
5823   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5824   rounded-corners .default:n = 4 pt
5825 }

```

The first argument of `\@@_hvlines_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).

```

5826 \cs_new_protected:Npn \@@_hvlines_block:nnn #1 #2 #3
5827 {
5828   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5829   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5830   \@@_cut_on_hyphen:w #2 \q_stop
5831   \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
5832   \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5833   \@@_cut_on_hyphen:w #3 \q_stop
5834   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5835   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5836   \pgfpicture
5837   \pgfrememberpicturepositiononpagetrue
5838   \pgf@relevantforpicturesizefalse
5839   \CT@arc@
5840   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

First, the vertical rules.

```

5841   \@@_qpoint:n { row - \l_tmpa_tl }
5842   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5843   \@@_qpoint:n { row - \l_tmpc_tl }
5844   \dim_set_eq:NN \l_tmpb_dim \pgf@y
5845   \int_step_inline:nnn \l_tmpd_tl \l_tmpb_tl
5846   {
5847     \@@_qpoint:n { col - ##1 }
5848     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpa_dim }
5849     \pgfpathlineto { \pgfpoint \pgf@x \l_tmpb_dim }
5850     \pgfusepathqstroke
5851   }

```

Now, the horizontal rules.

```

5852   \@@_qpoint:n { col - \l_tmpb_tl }
5853   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
5854   \@@_qpoint:n { col - \l_tmpd_tl }
5855   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \arrayrulewidth }
5856   \int_step_inline:nnn \l_tmpc_tl \l_tmpa_tl
5857   {
5858     \@@_qpoint:n { row - ##1 }
5859     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5860     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5861     \pgfusepathqstroke
5862   }
5863   \endpgfpicture
5864 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

5865 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
5866 {
5867   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
5868   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
5869   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
5870   { \@@_error:n { borders~forbidden } }
5871   {
5872     \clist_map_inline:Nn \l_@@_borders_clist
5873     {
5874       \clist_if_in:nnF { top , bottom , left , right } { ##1 }
5875       { \@@_error:nn { bad-border } { ##1 } }
5876     }
5877     \@@_cut_on_hyphen:w #2 \q_stop
5878     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl

```

```

5879 \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
5880 \@@_cut_on_hyphen:w #3 \q_stop
5881 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
5882 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
5883 \pgfpicture
5884 \pgfrememberpicturepositiononpagetrue
5885 \pgf@relevantforpicturesizefalse
5886 \CT@arc@
5887 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
5888 \clist_if_in:NnT \l_@@_borders_clist { right }
5889 { \@@_stroke_vertical:n \l_tmpb_tl }
5890 \clist_if_in:NnT \l_@@_borders_clist { left }
5891 { \@@_stroke_vertical:n \l_tmpd_tl }
5892 \clist_if_in:NnT \l_@@_borders_clist { bottom }
5893 { \@@_stroke_horizontal:n \l_tmpa_tl }
5894 \clist_if_in:NnT \l_@@_borders_clist { top }
5895 { \@@_stroke_horizontal:n \l_tmpc_tl }
5896 \endpgfpicture
5897 }
5898 }

```

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).

```

5899 \cs_new_protected:Npn \@@_stroke_vertical:n #1
5900 {
5901   \@@_qpoint:n \l_tmpc_tl
5902   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5903   \@@_qpoint:n \l_tmpa_tl
5904   \dim_set:Nn \l_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
5905   \@@_qpoint:n { #1 }
5906   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
5907   \pgfpathlineto { \pgfpoint \pgf@x \l_tmpc_dim }
5908   \pgfusepathqstroke
5909 }

```

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).

```

5910 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
5911 {
5912   \@@_qpoint:n \l_tmpd_tl
5913   \clist_if_in:NnTF \l_@@_borders_clist { left }
5914   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
5915   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
5916   \@@_qpoint:n \l_tmpb_tl
5917   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
5918   \@@_qpoint:n { #1 }
5919   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
5920   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
5921   \pgfusepathqstroke
5922 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

5923 \keys_define:nn { NiceMatrix / BlockBorders }
5924 {
5925   borders .clist_set:N = \l_@@_borders_clist ,
5926   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5927   rounded-corners .default:n = 4 pt ,
5928   line-width .dim_set:N = \l_@@_line_width_dim
5929 }

```

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.

```

5930 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
5931 {
5932   \begin { tikzpicture }
5933   \clist_map_inline:nn { #5 }
5934   {
5935     \path [ ##1 ]
5936       ( #1 -| #2 ) rectangle ( \@@_succ:n { #3 } -| \@@_succ:n { #4 } ) ;
5937   }
5938   \end { tikzpicture }
5939 }

```

How to draw the dotted lines transparently

```

5940 \cs_set_protected:Npn \@@_renew_matrix:
5941 {
5942   \RenewDocumentEnvironment { pmatrix } { } {
5943     { \pNiceMatrix }
5944     { \endpNiceMatrix }
5945   \RenewDocumentEnvironment { vmatrix } { } {
5946     { \vNiceMatrix }
5947     { \endvNiceMatrix }
5948   \RenewDocumentEnvironment { Vmatrix } { } {
5949     { \VNiceMatrix }
5950     { \endVNiceMatrix }
5951   \RenewDocumentEnvironment { bmatrix } { } {
5952     { \bNiceMatrix }
5953     { \endbNiceMatrix }
5954   \RenewDocumentEnvironment { Bmatrix } { } {
5955     { \BNiceMatrix }
5956     { \endBNiceMatrix }
5957 }

```

Automatic arrays

```

5958 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
5959 {
5960   \int_set:Nn \l_@@_nb_rows_int { #1 }
5961   \int_set:Nn \l_@@_nb_cols_int { #2 }
5962 }

```

We will extract the potential keys l, r and c and pass the other keys to the environment {NiceArrayWithDelims}.

```

5963 \keys_define:nn { NiceMatrix / Auto }
5964 {
5965   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
5966   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
5967   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
5968 }
5969 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
5970 {
5971   \int_zero_new:N \l_@@_nb_rows_int
5972   \int_zero_new:N \l_@@_nb_cols_int
5973   \@@_set_size:n #4 \q_stop

```

The group is for the protection of \l_@@_type_of_col_tl.

```

5974   \group_begin:
5975   \tl_set:Nn \l_@@_type_of_col_tl c
5976   \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
5977   \use:x
5978   {
5979     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
5980     { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }

```

```

5981     [ \exp_not:V \l_tmpa_tl ]
5982   }
5983   \int_compare:nNnT \l_@@_first_row_int = 0
5984   {
5985     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5986     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5987     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5988   }
5989   \prg_replicate:nn \l_@@_nb_rows_int
5990   {
5991     \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

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).

```

5992     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
5993     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
5994   }
5995   \int_compare:nNnT \l_@@_last_row_int > { -2 }
5996   {
5997     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
5998     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
5999     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6000   }
6001   \end { NiceArrayWithDelims }
6002   \group_end:
6003 }
6004 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6005 {
6006   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6007   {
6008     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6009     \AutoNiceMatrixWithDelims { #2 } { #3 }
6010   }
6011 }
6012 \@@_define_com:nnn p ( )
6013 \@@_define_com:nnn b [ ]
6014 \@@_define_com:nnn v | |
6015 \@@_define_com:nnn V \ | \ |
6016 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

6017 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
6018 {
6019   \group_begin:
6020   \bool_set_true:N \l_@@_NiceArray_bool
6021   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6022   \group_end:
6023 }

```

The redefinition of the command \dotfill

```

6024 \cs_set_eq:NN \@@_old_dotfill \dotfill
6025 \cs_new_protected:Npn \@@_dotfill:
6026 {

```

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}).

```

6027   \@@_old_dotfill
6028   \bool_if:NT \l_@@_NiceTabular_bool
6029   { \group_insert_after:N \@@_dotfill_ii: }
6030   { \group_insert_after:N \@@_dotfill_i: }
6031 }

```

```

6032 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6033 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

```

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 its 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`.

```

6034 \cs_new_protected:Npn \@@_dotfill_iii:
6035 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim { \@@_old_dotfill }

```

The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`.

```

6036 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6037 {
6038   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6039   {
6040     \@@_actually_diagbox:nnnnnn
6041     { \int_use:N \c_iRow }
6042     { \int_use:N \c_jCol }
6043     { \int_use:N \c_iRow }
6044     { \int_use:N \c_jCol }
6045     { \exp_not:n { #1 } }
6046     { \exp_not:n { #2 } }
6047   }

```

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`.

```

6048   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6049   {
6050     { \int_use:N \c_iRow }
6051     { \int_use:N \c_jCol }
6052     { \int_use:N \c_iRow }
6053     { \int_use:N \c_jCol }
6054   }
6055 }

```

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

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The two other are the elements to draw below and above the diagonal line.

```

6056 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
6057 {
6058   \pgfpicture
6059   \pgf@relevantforpicturesizefalse
6060   \pgfrememberpicturepositiononpagetrue
6061   \@@_qpoint:n { row - #1 }
6062   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6063   \@@_qpoint:n { col - #2 }
6064   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6065   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6066   \@@_qpoint:n { row - \@@_succ:n { #3 } }
6067   \dim_set_eq:NN \l_tmpc_dim \pgf@y
6068   \@@_qpoint:n { col - \@@_succ:n { #4 } }
6069   \dim_set_eq:NN \l_tmpd_dim \pgf@x
6070   \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
6071   {

```

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.

```

6072   \CT@arc@
6073   \pgfsetroundcap
6074   \pgfusepathqstroke
6075 }

```

```

6076 \pgfset { inner-sep = 1 pt }
6077 \pgfscope
6078 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
6079 \pgfnode { rectangle } { south-west }
6080 { \@@_math_toggle_token: #5 \@@_math_toggle_token: } { } { }
6081 \endpgfscope
6082 \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
6083 \pgfnode { rectangle } { north-east }
6084 { \@@_math_toggle_token: #6 \@@_math_toggle_token: } { } { }
6085 \endpgfpicture
6086 }

```

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.

```

6087 \keys_define:nn { NiceMatrix }
6088 {
6089   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6090   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6091 }
6092 \keys_define:nn { NiceMatrix / CodeAfter }
6093 {
6094   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6095   sub-matrix .value_required:n = true ,
6096   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6097   delimiters / color .value_required:n = true ,
6098   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6099   rules .value_required:n = true ,
6100   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
6101 }

```

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. 110.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6102 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which do *not* begin with `\omit` (and thus, the user will be able to use `\CodeAfter` without error and without the need to prefix by `\omit`).

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

6103 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
6104 {
6105   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6106   \@@_CodeAfter_ii:n
6107 }

```

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

```

6108 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1
6109 {

```

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.

```

6110   \str_if_eq:eeTF \@currenvir { #1 } { \end { #1 } }

```

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`.

```

6111 {
6112   \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6113   \@@_CodeAfter_i:n
6114 }
6115 }
```

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 (`(`, `[`, `\{`, `)`, `]` or `\}`). 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.

```

6116 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6117 {
6118   \pgfpicture
6119   \pgfrememberpicturepositiononpagetrue
6120   \pgf@relevantforpicturesizefalse
```

`\l_@@_y_initial_dim` and `\l_@@_y_final_dim` will be the y -values of the extremities of the delimiter we will have to construct.

```

6121 \@@_qpoint:n { row - 1 }
6122 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6123 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
6124 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
```

We will compute in `\l_tmpa_dim` the x -value where we will have to put our delimiter (on the left side or on the right side).

```

6125 \bool_if:nTF { #3 }
6126 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6127 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6128 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6129 {
6130   \cs_if_exist:cT
6131   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6132   {
6133     \pgfpointanchor
6134     { \@@_env: - ##1 - #2 }
6135     { \bool_if:nTF { #3 } { west } { east } }
6136     \dim_set:Nn \l_tmpa_dim
6137     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6138   }
6139 }
```

Now we can put the delimiter with a node of PGF.

```

6140 \pgfset { inner~sep = \c_zero_dim }
6141 \dim_zero:N \nulldelimiterspace
6142 \pgftransformshift
6143 {
6144   \pgfpoint
6145   { \l_tmpa_dim }
6146   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6147 }
6148 \pgfnode
```

```

6149 { rectangle }
6150 { \bool_if:nTF { #3 } { east } { west } }
6151 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6152 \nullfont
6153 \c_math_toggle_token
6154 \tl_if_empty:NF \l_@@_delimiters_color_tl
6155 { \color { \l_@@_delimiters_color_tl } }
6156 \bool_if:nTF { #3 } { \left #1 } { \left . }
6157 \vcenter
6158 {
6159 \nullfont
6160 \hrule \@height
6161 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6162 \@depth \c_zero_dim
6163 \@width \c_zero_dim
6164 }
6165 \bool_if:nTF { #3 } { \right . } { \right #1 }
6166 \c_math_toggle_token
6167 }
6168 { }
6169 { }
6170 \endpgfpicture
6171 }

```

The command \SubMatrix

```

6172 \keys_define:nn { NiceMatrix / sub-matrix }
6173 {
6174 extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6175 extra-height .value_required:n = true ,
6176 left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6177 left-xshift .value_required:n = true ,
6178 right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6179 right-xshift .value_required:n = true ,
6180 xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6181 xshift .value_required:n = true ,
6182 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6183 delimiters / color .value_required:n = true ,
6184 slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6185 slim .default:n = true ,
6186 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6187 hlines .default:n = all ,
6188 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6189 vlines .default:n = all ,
6190 hvlines .meta:n = { hlines, vlines } ,
6191 hvlines .value_forbidden:n = true ,
6192 }
6193 \keys_define:nn { NiceMatrix }
6194 {
6195 SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6196 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6197 NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6198 NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6199 pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6200 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6201 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

6202 \keys_define:nn { NiceMatrix / SubMatrix }
6203 {

```



```

6204 hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6205 hlines .default:n = all ,
6206 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6207 vlines .default:n = all ,
6208 hvlines .meta:n = { hlines, vlines } ,
6209 hvlines .value_forbidden:n = true ,
6210 name .code:n =
6211   \tl_if_empty:nTF { #1 }
6212   { \@@_error:n { Invalid~name~format } }
6213   {
6214     \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6215     {
6216       \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6217       { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
6218       {
6219         \str_set:Nn \l_@@_submatrix_name_str { #1 }
6220         \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6221       }
6222     }
6223     { \@@_error:n { Invalid~name~format } }
6224   } ,
6225 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6226 rules .value_required:n = true ,
6227 code .tl_set:N = \l_@@_code_tl ,
6228 code .value_required:n = true ,
6229 name .value_required:n = true ,
6230 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
6231 }

6232 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
6233 {
6234   \peek_remove_spaces:n
6235   {
6236     \@@_cut_on_hyphen:w #3 \q_stop
6237     \tl_clear_new:N \l_tmpc_tl
6238     \tl_clear_new:N \l_tmpd_tl
6239     \tl_set_eq:NN \l_tmpc_tl \l_tmpa_tl
6240     \tl_set_eq:NN \l_tmpd_tl \l_tmpb_tl
6241     \@@_cut_on_hyphen:w #2 \q_stop
6242     \seq_gput_right:Nx \g_@@_submatrix_seq
6243     { { \l_tmpa_tl } { \l_tmpb_tl } { \l_tmpc_tl } { \l_tmpd_tl } }
6244     \tl_gput_right:Nn \g_@@_internal_code_after_tl
6245     { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6246   }
6247 }

```

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`.

```

6248 \AtBeginDocument
6249 {
6250   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6251   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6252   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6253     {
6254       \peek_remove_spaces:n
6255       { \@@_sub_matrix:nnnnnnn
6256         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
6257     }
6258 }

6259 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6260 {
6261   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

6262   \tl_clear_new:N \l_@@_first_i_tl
6263   \tl_clear_new:N \l_@@_first_j_tl
6264   \tl_clear_new:N \l_@@_last_i_tl
6265   \tl_clear_new:N \l_@@_last_j_tl

```

The command `\@@_cut_on_hyphen:w` cuts on the hyphen an argument of the form $i-j$. The value of i is stored in `\l_tmpa_tl` and the value of j is stored in `\l_tmpb_tl`.

```

6266   \@@_cut_on_hyphen:w #2 \q_stop
6267   \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl
6268   \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl
6269   \@@_cut_on_hyphen:w #3 \q_stop
6270   \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl
6271   \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl
6272   \bool_lazy_or:nnTF
6273     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6274     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6275     { \@@_error:n { SubMatrix~too~large } }
6276   {
6277     \str_clear_new:N \l_@@_submatrix_name_str
6278     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6279     \pgfpicture
6280     \pgfrememberpicturerepositiononpagetrue
6281     \pgf@relevantforpicturesizefalse
6282     \pgfset { inner-sep = \c_zero_dim }
6283     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6284     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

6285   \bool_if:NTF \l_@@_submatrix_slim_bool
6286     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6287     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6288   {
6289     \cs_if_exist:cT
6290       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6291     {
6292       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6293       \dim_set:Nn \l_@@_x_initial_dim
6294         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6295     }
6296     \cs_if_exist:cT
6297       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6298     {
6299       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6300       \dim_set:Nn \l_@@_x_final_dim
6301         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6302     }

```

```

6303     }
6304     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6305     { \@@_error:nn { impossible~delimiter } { left } }
6306     {
6307         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6308         { \@@_error:nn { impossible~delimiter } { right } }
6309         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6310     }
6311     \endpgfpicture
6312 }
6313 \group_end:
6314 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

6315 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6316 {
6317     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6318     \dim_set:Nn \l_@@_y_initial_dim
6319     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6320     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6321     \dim_set:Nn \l_@@_y_final_dim
6322     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6323     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6324     {
6325         \cs_if_exist:cT
6326         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6327         {
6328             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6329             \dim_set:Nn \l_@@_y_initial_dim
6330             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6331         }
6332         \cs_if_exist:cT
6333         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6334         {
6335             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6336             \dim_set:Nn \l_@@_y_final_dim
6337             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6338         }
6339     }
6340     \dim_set:Nn \l_tmpa_dim
6341     {
6342         \l_@@_y_initial_dim - \l_@@_y_final_dim +
6343         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6344     }
6345     \dim_set_eq:NN \nulldelimiterspace \c_zero_dim

```

We will draw the rules in the \SubMatrix.

```

6346 \group_begin:
6347 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6348 \tl_if_empty:NF \l_@@_rules_color_tl
6349 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6350 \CT@arc@

```

Now, we draw the potential vertical rules specified in the preamble of the environments with the letter fixed with the key `vlines-in-sub-matrix`. The list of the columns where there is such rule to draw is in `\g_@@_cols_vlism_seq`.

```

6351 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6352 {
6353     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6354     {
6355         \int_compare:nNnT
6356         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6357         {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

6358         \@@_qpoint:n { col - ##1 }
6359         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6360         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6361         \pgfusepathqstroke
6362     }
6363 }
6364 }

```

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.

```

6365     \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6366     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6367     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6368     {
6369         \bool_lazy_and:nnTF
6370         { \int_compare_p:nNn { ##1 } > 0 }
6371         {
6372             \int_compare_p:nNn
6373             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6374         {
6375             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6376             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6377             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6378             \pgfusepathqstroke
6379         }
6380         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
6381     }

```

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.

```

6382     \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6383     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6384     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6385     {
6386         \bool_lazy_and:nnTF
6387         { \int_compare_p:nNn { ##1 } > 0 }
6388         {
6389             \int_compare_p:nNn
6390             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6391         {
6392             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6393     \group_begin:
6394     \dim_set:Nn \l_tmpa_dim
6395     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6396     \str_case:nn { #1 }
6397     {
6398         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6399         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6400         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6401     }
6402     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

6403     \dim_set:Nn \l_tmpb_dim
6404     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6405     \str_case:nn { #2 }
6406     {
6407         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6408         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }

```

```

6409         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6410     }
6411     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6412     \pgfusepathqstroke
6413     \group_end:
6414 }
6415 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
6416 }

```

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).

```

6417 \str_if_empty:NF \l_@@_submatrix_name_str
6418 {
6419     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6420     \l_@@_x_initial_dim \l_@@_y_initial_dim
6421     \l_@@_x_final_dim \l_@@_y_final_dim
6422 }
6423 \group_end:

```

The group was for `\CT@arc@` (the color of the rules).

Now, we deal with the left delimiter. Of course, the environment `{pgfscope}` is for the `\pgftransformshift`.

```

6424 \begin { pgfscope }
6425 \pgftransformshift
6426 {
6427     \pgfpoint
6428     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6429     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6430 }
6431 \str_if_empty:NTF \l_@@_submatrix_name_str
6432 { \@@_node_left:nn #1 { } }
6433 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6434 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6435 \pgftransformshift
6436 {
6437     \pgfpoint
6438     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6439     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6440 }
6441 \str_if_empty:NTF \l_@@_submatrix_name_str
6442 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
6443 {
6444     \@@_node_right:nnnn #2
6445     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
6446 }
6447 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
6448 \flag_clear_new:n { nicematrix }
6449 \l_@@_code_tl
6450 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

6451 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #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.

```

6452 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
6453 {
6454   \use:e
6455   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpntanchor_i:nn #1 } }
6456 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\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 -.

```

6457 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
6458 { #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.

```

6459 \tl_const:Nn \c_@@_integers_alist_tl
6460 {
6461   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
6462   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
6463   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
6464   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
6465 }

```

```

6466 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
6467 {

```

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 $i-|j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row of a number of column, we keep track of the number of such treatments by the expandable flag called **nicematrix**.

```

6468   \tl_if_empty:nTF { #2 }
6469   {
6470     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
6471     {
6472       \flag_raise:n { nicematrix }
6473       \int_if_even:nTF { \flag_height:n { nicematrix } }
6474       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
6475       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
6476     }
6477     { #1 }
6478   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

6479     { \@@_pgfpointanchor_iii:w { #1 } #2 }
6480   }

```

There was an hyphen in the name of the node and that’s why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

6481 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
6482 {
6483   \str_case:nnF { #1 }
6484   {
6485     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
6486     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
6487   }

```

Now the case of a node of the form $i-j$.

```

6488     {
6489         \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
6490         - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
6491     }
6492 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

6493 \cs_new_protected:Npn \@@_node_left:nn #1 #2
6494 {
6495     \pgfnode
6496     { rectangle }
6497     { east }
6498     {
6499         \nullfont
6500         \c_math_toggle_token
6501         \tl_if_empty:NF \l_@@_delimiters_color_tl
6502         { \color { \l_@@_delimiters_color_tl } }
6503         \left #1
6504         \vcenter
6505         {
6506             \nullfont
6507             \hrule \@height \l_tmpa_dim
6508                 \@depth \c_zero_dim
6509                 \@width \c_zero_dim
6510         }
6511         \right .
6512         \c_math_toggle_token
6513     }
6514     { #2 }
6515     { }
6516 }

```

The command `\@@_node_right:nnn` 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.

```

6517 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
6518 {
6519     \pgfnode
6520     { rectangle }
6521     { west }
6522     {
6523         \nullfont
6524         \c_math_toggle_token
6525         \tl_if_empty:NF \l_@@_delimiters_color_tl
6526         { \color { \l_@@_delimiters_color_tl } }
6527         \left .
6528         \vcenter
6529         {
6530             \nullfont
6531             \hrule \@height \l_tmpa_dim
6532                 \@depth \c_zero_dim
6533                 \@width \c_zero_dim
6534         }
6535         \right #1
6536         \tl_if_empty:NF { #3 } { _ { \smash { #3 } } }
6537         ^ { \smash { #4 } }
6538         \c_math_toggle_token
6539     }
6540     { #2 }
6541     { }

```

```
6542 }
```

We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```
6543 \bool_new:N \c_@@_footnotehyper_bool
```

The boolean `\c_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to true if the option `footnotehyper` is used.

```
6544 \bool_new:N \c_@@_footnote_bool
6545 \@@_msg_new:nnn { Unknown-key~for~package }
6546 {
6547   The~key~'\l_keys_key_str'~is~unknown. \\\
6548   If~you~go~on,~it~will~be~ignored. \\\
6549   For~a~list~of~the~available~keys,~type~H~<return>.
6550 }
6551 {
6552   The~available~keys~are~(in~alphabetic~order):~
6553   define-L-C-R,~
6554   footnote,~
6555   footnotehyper,~
6556   renew-dots,~and
6557   renew-matrix.
6558 }
```

Maybe we will completely delete the key 'transparent' in a future version.

```
6559 \@@_msg_new:nn { Key~transparent }
6560 {
6561   The~key~'transparent'~is~now~obsolete~(because~it's~name~
6562   is~not~clear).~You~should~use~the~conjunction~of~'renew-dots'~
6563   and~'renew-matrix'.~However,~you~can~go~on.
6564 }
6565 \@@_msg_new:nn { define-L-C-R }
6566 {
6567   You~have~used~the~key~'define-L-C-R'.~Please~note~that~this~key~
6568   will~probably~be~deleted~in~a~future~version~of~'nicematrix'.\\
6569   However,~you~can~go~on.
6570 }
6571 \keys_define:nn { NiceMatrix / Package }
6572 {
6573   define-L-C-R .code:n =
6574     \@@_error:n { define-L-C-R }
6575     \bool_set_true:N \c_@@_define_L_C_R_bool ,
6576   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
6577   renew-dots .value_forbidden:n = true ,
6578   renew-matrix .code:n = \@@_renew_matrix: ,
6579   renew-matrix .value_forbidden:n = true ,
6580   transparent .code:n =
6581     {
6582       \@@_renew_matrix:
6583       \bool_set_true:N \l_@@_renew_dots_bool
6584       \@@_error:n { Key~transparent }
6585     } ,
```



```

6586     transparent .value_forbidden:n = true,
6587     footnote .bool_set:N = \c_@@_footnote_bool ,
6588     footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
6589     unknown .code:n = \@@_error:n { Unknown~key~for~package }
6590   }
6591   \ProcessKeysOptions { NiceMatrix / Package }

6592   \@@_msg_new:nn { footnote~with~footnotehyper~package }
6593   {
6594     You~can't~use~the~option~'footnote'~because~the~package~
6595     footnotehyper~has~already~been~loaded.~
6596     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
6597     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6598     of~the~package~footnotehyper.\\
6599     If~you~go~on,~the~package~footnote~won't~be~loaded.
6600   }

6601   \@@_msg_new:nn { footnotehyper~with~footnote~package }
6602   {
6603     You~can't~use~the~option~'footnotehyper'~because~the~package~
6604     footnote~has~already~been~loaded.~
6605     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
6606     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
6607     of~the~package~footnote.\\
6608     If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
6609   }

6610   \bool_if:NT \c_@@_footnote_bool
6611   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6612     \@ifclassloaded { beamer }
6613     { \bool_set_false:N \c_@@_footnote_bool }
6614     {
6615       \@ifpackageloaded { footnotehyper }
6616       { \@@_error:n { footnote~with~footnotehyper~package } }
6617       { \usepackage { footnote } }
6618     }
6619   }

6620   \bool_if:NT \c_@@_footnotehyper_bool
6621   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

6622     \@ifclassloaded { beamer }
6623     { \bool_set_false:N \c_@@_footnote_bool }
6624     {
6625       \@ifpackageloaded { footnote }
6626       { \@@_error:n { footnotehyper~with~footnote~package } }
6627       { \usepackage { footnotehyper } }
6628     }
6629     \bool_set_true:N \c_@@_footnote_bool
6630   }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

The following message will be deleted when we will delete the key `except-corners` for the command `\arraycolor`.

```

6631 \@@_msg_new:nn { key except-corners }
6632 {
6633   The~key~'except-corners'~has-been-deleted-for~the~command~\token_to_str:N
6634   \arraycolor\ in~the~\token_to_str:N \CodeBefore.~You~should~instead~use~
6635   the~key~'corners'~in~your~\@@_full_name_env:.\
6636   If~you~go~on,~this~key~will~be~ignored.
6637 }
6638 \seq_new:N \c_@@_types_of_matrix_seq
6639 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
6640 {
6641   NiceMatrix ,
6642   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
6643 }
6644 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
6645 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTf` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

6646 \cs_new_protected:Npn \@@_error_too_much_cols:
6647 {
6648   \seq_if_in:NVTf \c_@@_types_of_matrix_seq \g_@@_name_env_str
6649   {
6650     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
6651     { \@@_fatal:n { too-much-cols-for-matrix } }
6652     {
6653       \bool_if:NF \l_@@_last_col_without_value_bool
6654       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
6655     }
6656   }
6657   { \@@_fatal:n { too-much-cols-for-array } }
6658 }

```

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

```

6659 \cs_new:Npn \@@_message_hdotsfor:
6660 {
6661   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
6662   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
6663 }
6664 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
6665 {
6666   You~try~to~use~more~columns~than~allowed~by~your~
6667   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
6668   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
6669   exterior~columns).~This~error~is~fatal.
6670 }
6671 \@@_msg_new:nn { too-much-cols-for-matrix }
6672 {
6673   You~try~to~use~more~columns~than~allowed~by~your~
6674   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
6675   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
6676   'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
6677   This~error~is~fatal.
6678 }

```

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.

```

6679 \@@_msg_new:nn { too-much-cols-for-array }
6680 {
6681   You-try-to-use-more-columns-than-allowed-by-your-
6682   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
6683   \int_use:N \g_@@_static_num_of_col_int\
6684   ~(plus-the-potential-exterior-ones).~
6685   This-error-is-fatal.
6686 }
6687 \@@_msg_new:nn { last-col-not-used }
6688 {
6689   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
6690   in~your~\@@_full_name_env:~.~However,~you~can~go~on.
6691 }
6692 \@@_msg_new:nn { columns-not-used }
6693 {
6694   The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6695   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
6696   You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
6697 }
6698 \@@_msg_new:nn { in-first-col }
6699 {
6700   You~can't~use~the~command~\#1 in~the~first~column~(number~0)~of~the~array.\
6701   If~you~go~on,~this~command~will~be~ignored.
6702 }
6703 \@@_msg_new:nn { in-last-col }
6704 {
6705   You~can't~use~the~command~\#1 in~the~last~column~(exterior)~of~the~array.\
6706   If~you~go~on,~this~command~will~be~ignored.
6707 }
6708 \@@_msg_new:nn { in-first-row }
6709 {
6710   You~can't~use~the~command~\#1 in~the~first~row~(number~0)~of~the~array.\
6711   If~you~go~on,~this~command~will~be~ignored.
6712 }
6713 \@@_msg_new:nn { in-last-row }
6714 {
6715   You~can't~use~the~command~\#1 in~the~last~row~(exterior)~of~the~array.\
6716   If~you~go~on,~this~command~will~be~ignored.
6717 }
6718 \@@_msg_new:nn { double-closing-delimiter }
6719 {
6720   You~can't~put~a~second~closing~delimiter~\#1~just~after~a~first~closing~
6721   delimiter.~This~delimiter~will~be~ignored.
6722 }
6723 \@@_msg_new:nn { delimiter-after-opening }
6724 {
6725   You~can't~put~a~second~delimiter~\#1~just~after~a~first~opening~
6726   delimiter.~This~delimiter~will~be~ignored.
6727 }
6728 \@@_msg_new:nn { bad-option-for-line-style }
6729 {
6730   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
6731   is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
6732 }
6733 \@@_msg_new:nn { Unknown-key-for-xdots }
6734 {
6735   As~for~now,~there~is~only~three~keys~available~here:~'color',~'line-style'~

```

```

6736     and~'shorten'~(and~you~try~to~use~'\l_keys_key_str')~.~If~you~go~on,~
6737     this~key~will~be~ignored.
6738 }

6739 \@@_msg_new:nn { Unknown~key~for~RowStyle }
6740 {
6741     As~for~now,~there~is~only~three~keys~available~here:~'cell-space-top-limit',~
6742     'cell-space-bottom-limit~and~'cell-space-limits'~(and~you~try~to~use~
6743     '\l_keys_key_str')~.~If~you~go~on,~this~key~will~be~ignored.
6744 }

6745 \@@_msg_new:nn { Unknown~key~for~rowcolors }
6746 {
6747     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
6748     (and~you~try~to~use~'\l_keys_key_str')~.~If~you~go~on,~
6749     this~key~will~be~ignored.
6750 }

6751 \@@_msg_new:nn { ampersand~in~light-syntax }
6752 {
6753     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
6754     ~you~have~used~the~key~'light-syntax'~.~This~error~is~fatal.
6755 }

6756 \@@_msg_new:nn { SubMatrix~too~large }
6757 {
6758     Your~command~\token_to_str:N \SubMatrix\
6759     can't~be~drawn~because~your~matrix~is~too~small.\\
6760     If~you~go~on,~this~command~will~be~ignored.
6761 }

6762 \@@_msg_new:nn { double-backslash~in~light-syntax }
6763 {
6764     You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
6765     the~key~'light-syntax'~.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
6766     (set~by~the~key~'end-of-row')~.~This~error~is~fatal.
6767 }

6768 \@@_msg_new:nn { standard~cline~in~document }
6769 {
6770     The~key~'standard~cline'~is~available~only~in~the~preamble.\\
6771     If~you~go~on~this~command~will~be~ignored.
6772 }

6773 \@@_msg_new:nn { old~column~type }
6774 {
6775     The~column~type~'#1'~is~no~longer~defined~in~'nicematrix'~.~
6776     Since~version~5.0,~you~have~to~use~'l',~'c'~and~'r'~instead~of~'L',~
6777     'C'~and~'R'~.~You~can~also~use~the~key~'define-L-C-R'~.\\
6778     This~error~is~fatal.
6779 }

6780 \@@_msg_new:nn { bad~value~for~baseline }
6781 {
6782     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
6783     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
6784     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
6785     If~you~go~on,~a~value~of~1~will~be~used.
6786 }

6787 \@@_msg_new:nn { Invalid~name~format }
6788 {
6789     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
6790     \SubMatrix.\\
6791     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
6792     If~you~go~on,~this~key~will~be~ignored.
6793 }

6794 \@@_msg_new:nn { Wrong~line~in~SubMatrix }

```

```

6795 {
6796     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
6797     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
6798     number~is~not~valid.~If~you~go~on,~it~will~be~ignored.
6799 }
6800 \@@_msg_new:nn { impossible-delimiter }
6801 {
6802     It's~impossible~to~draw~the~#1~delimiter~of~your~
6803     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
6804     in~that~column.
6805     \bool_if:NT \l_@@_submatrix_slim_bool
6806     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
6807     If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
6808 }
6809 \@@_msg_new:nn { empty-environment }
6810 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
6811 \@@_msg_new:nn { Delimiter~with~small }
6812 {
6813     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
6814     because~the~key~'small'~is~in~force.\\
6815     This~error~is~fatal.
6816 }
6817 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
6818 {
6819     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
6820     can't~be~executed~because~a~cell~doesn't~exist.\\
6821     If~you~go~on~this~command~will~be~ignored.
6822 }
6823 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
6824 {
6825     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
6826     in~this~\@@_full_name_env:.\
6827     If~you~go~on,~this~key~will~be~ignored.\\
6828     For~a~list~of~the~names~already~used,~type~H~<return>.
6829 }
6830 {
6831     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
6832     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
6833 }
6834 \@@_msg_new:nn { r-or-l~with~preamble }
6835 {
6836     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
6837     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
6838     your~\@@_full_name_env:.\
6839     If~you~go~on,~this~key~will~be~ignored.
6840 }
6841 \@@_msg_new:nn { Hdotsfor~in~col-0 }
6842 {
6843     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
6844     the~array.~This~error~is~fatal.
6845 }
6846 \@@_msg_new:nn { bad~corner }
6847 {
6848     #1~is~an~incorrect~specification~for~a~corner~(in~the~keys~
6849     'corners'~and~'except-corners').~The~available~
6850     values~are::~NW,~SW,~NE~and~SE.\\
6851     If~you~go~on,~this~specification~of~corner~will~be~ignored.
6852 }
6853 \@@_msg_new:nn { bad~border }
6854 {

```

```

6855     #1~is~an~incorrect~specification~for~a~border~(in~the~key~
6856     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
6857     values~are:~left,~right,~top~and~bottom.\\
6858     If~you~go~on,~this~specification~of~border~will~be~ignored.
6859 }

6860 \@@_msg_new:nn { tikz~key~without~tikz }
6861 {
6862     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
6863     \Block'~because~you~have~not~loaded~Tikz.~
6864     If~you~go~on,~this~key~will~be~ignored.
6865 }

6866 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
6867 {
6868     In~the~\@@_full_name_env:,~you~must~use~the~key~
6869     'last~col'~without~value.\\
6870     However,~you~can~go~on~for~this~time~
6871     (the~value~'\l_keys_value_tl'~will~be~ignored).
6872 }

6873 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
6874 {
6875     In~\NiceMatrixoptions,~you~must~use~the~key~
6876     'last~col'~without~value.\\
6877     However,~you~can~go~on~for~this~time~
6878     (the~value~'\l_keys_value_tl'~will~be~ignored).
6879 }

6880 \@@_msg_new:nn { Block~too~large~1 }
6881 {
6882     You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
6883     too~small~for~that~block. \\
6884 }

6885 \@@_msg_new:nn { Block~too~large~2 }
6886 {
6887     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
6888     \g_@@_static_num_of_col_int\
6889     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
6890     specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
6891     (&)~at~the~end~of~the~first~row~of~your~
6892     \@@_full_name_env:.\
6893     If~you~go~on,~this~block~and~maybe~others~will~be~ignored.
6894 }

6895 \@@_msg_new:nn { unknown~column~type }
6896 {
6897     The~column~type~'#1'~in~your~\@@_full_name_env:\
6898     is~unknown. \\
6899     This~error~is~fatal.
6900 }

6901 \@@_msg_new:nn { tabularnote~forbidden }
6902 {
6903     You~can't~use~the~command~\token_to_str:N\ tabularnote\
6904     ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
6905     \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
6906     If~you~go~on,~this~command~will~be~ignored.
6907 }

6908 \@@_msg_new:nn { borders~forbidden }
6909 {
6910     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
6911     because~the~option~'rounded~corners'~
6912     is~in~force~with~a~non~zero~value.\\
6913     If~you~go~on,~this~key~will~be~ignored.
6914 }

```

```

6915 \@@_msg_new:nn { bottomrule-without-booktabs }
6916 {
6917   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
6918   loaded~'booktabs'.\\
6919   If~you~go~on,~this~key~will~be~ignored.
6920 }
6921 \@@_msg_new:nn { enumitem~not~loaded }
6922 {
6923   You~can't~use~the~command~\token_to_str:N\tabularnote\
6924   ~because~you~haven't~loaded~'enumitem'.\\
6925   If~you~go~on,~this~command~will~be~ignored.
6926 }
6927 \@@_msg_new:nn { Wrong~last~row }
6928 {
6929   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
6930   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
6931   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
6932   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
6933   without~value~(more~compilations~might~be~necessary).
6934 }
6935 \@@_msg_new:nn { Yet~in~env }
6936 { Environments~of~nicematrix~can't~be~nested.\\ This~error~is~fatal. }
6937 \@@_msg_new:nn { Outside~math~mode }
6938 {
6939   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
6940   (and~not~in~\token_to_str:N \vcenter).\\
6941   This~error~is~fatal.
6942 }
6943 \@@_msg_new:nn { One~letter~allowed }
6944 {
6945   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
6946   If~you~go~on,~it~will~be~ignored.
6947 }
6948 \@@_msg_new:nnn { Unknown~key~for~Block }
6949 {
6950   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
6951   \Block.\\ If~you~go~on,~it~will~be~ignored. \\
6952   For~a~list~of~the~available~keys,~type~H~<return>.
6953 }
6954 {
6955   The~available~keys~are~(in~alphabetic~order):~b,~borders,~c,~draw,~fill,~
6956   hvlines,~l,~line~width,~rounded~corners,~r,~t~and~tikz.
6957 }
6958 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
6959 {
6960   The~key~'\l_keys_key_str'~is~unknown.\\
6961   If~you~go~on,~it~will~be~ignored. \\
6962   For~a~list~of~the~available~keys~in~\token_to_str:N
6963   \CodeAfter,~type~H~<return>.
6964 }
6965 {
6966   The~available~keys~are~(in~alphabetic~order):~
6967   delimiters/color,~
6968   rules~(with~the~subkeys~'color'~and~'width'),~
6969   sub~matrix~(several~subkeys)~
6970   and~xdots~(several~subkeys).~
6971   The~latter~is~for~the~command~\token_to_str:N \line.
6972 }
6973 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
6974 {

```

```

6975 The~key~'\l_keys_key_str'~is~unknown.\\
6976 If~you~go~on,~this~key~will~be~ignored. \\
6977 For~a~list~of~the~available~keys~in~\token_to_str:N
6978 \SubMatrix,~type~H~<return>.
6979 }
6980 {
6981 The~available~keys~are~(in~alphabetic~order):~
6982 'delimiters/color',~
6983 'extra-height',~
6984 'hlines',~
6985 'hvlines',~
6986 'left-xshift',~
6987 'name',~
6988 'right-xshift',~
6989 'rules'~(with~the~subkeys~'color'~and~'width'),~
6990 'slim',~
6991 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
6992 and~'right-xshift').\\
6993 }
6994 \@@_msg_new:nnn { Unknown~key~for~notes }
6995 {
6996 The~key~'\l_keys_key_str'~is~unknown.\\
6997 If~you~go~on,~it~will~be~ignored. \\
6998 For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
6999 }
7000 {
7001 The~available~keys~are~(in~alphabetic~order):~
7002 bottomrule,~
7003 code-after,~
7004 code-before,~
7005 enumitem-keys,~
7006 enumitem-keys-para,~
7007 para,~
7008 label-in-list,~
7009 label-in-tabular~and~
7010 style.
7011 }
7012 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7013 {
7014 The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7015 \token_to_str:N \NiceMatrixOptions. \\
7016 If~you~go~on,~it~will~be~ignored. \\
7017 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7018 }
7019 {
7020 The~available~keys~are~(in~alphabetic~order):~
7021 allow-duplicate-names,~
7022 cell-space-bottom-limit,~
7023 cell-space-limits,~
7024 cell-space-top-limit,~
7025 code-for-first-col,~
7026 code-for-first-row,~
7027 code-for-last-col,~
7028 code-for-last-row,~
7029 corners,~
7030 create-extra-nodes,~
7031 create-medium-nodes,~
7032 create-large-nodes,~
7033 delimiters~(several~subkeys),~
7034 end-of-row,~
7035 first-col,~
7036 first-row,~
7037 hlines,~

```



```

7038     hvlines,~
7039     last-col,~
7040     last-row,~
7041     left-margin,~
7042     letter-for-dotted-lines,~
7043     light-syntax,~
7044     notes~(several~subkeys),~
7045     nullify-dots,~
7046     renew-dots,~
7047     renew-matrix,~
7048     right-margin,~
7049     rules~(with~the~subkeys~'color'~and~'width'),~
7050     small,~
7051     sub-matrix~(several~subkeys),
7052     vlines,~
7053     xdots~(several~subkeys).
7054 }

7055 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
7056 {
7057     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7058     \{NiceArray\}. \\
7059     If~you~go~on,~it~will~be~ignored. \\
7060     For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7061 }
7062 {
7063     The~available~keys~are~(in~alphabetic~order):~
7064     b,~
7065     baseline,~
7066     c,~
7067     cell-space-bottom-limit,~
7068     cell-space-limits,~
7069     cell-space-top-limit,~
7070     code-after,~
7071     code-for-first-col,~
7072     code-for-first-row,~
7073     code-for-last-col,~
7074     code-for-last-row,~
7075     colortbl-like,~
7076     columns-width,~
7077     corners,~
7078     create-extra-nodes,~
7079     create-medium-nodes,~
7080     create-large-nodes,~
7081     delimiters/color,~
7082     extra-left-margin,~
7083     extra-right-margin,~
7084     first-col,~
7085     first-row,~
7086     hlines,~
7087     hvlines,~
7088     last-col,~
7089     last-row,~
7090     left-margin,~
7091     light-syntax,~
7092     name,~
7093     notes/bottomrule,~
7094     notes/para,~
7095     nullify-dots,~
7096     renew-dots,~
7097     right-margin,~
7098     rules~(with~the~subkeys~'color'~and~'width'),~
7099     small,~
7100     t,~

```

```

7101     tabularnote,~
7102     vl原因,~
7103     xdots/color,~
7104     xdots/shorten~and~
7105     xdots/line-style.
7106 }

```

This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is also the keys t, c and b).

```

7107 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
7108 {
7109   The~key~'\l_keys_key_str'~is~unknown~for~the~
7110   \@@_full_name_env:. \\\
7111   If~you~go~on,~it~will~be~ignored. \\\
7112   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7113 }
7114 {
7115   The~available~keys~are~(in~alphabetic~order):~
7116   b,~
7117   baseline,~
7118   c,~
7119   cell-space-bottom-limit,~
7120   cell-space-limits,~
7121   cell-space-top-limit,~
7122   code-after,~
7123   code-for-first-col,~
7124   code-for-first-row,~
7125   code-for-last-col,~
7126   code-for-last-row,~
7127   colortbl-like,~
7128   columns-width,~
7129   corners,~
7130   create-extra-nodes,~
7131   create-medium-nodes,~
7132   create-large-nodes,~
7133   delimiters~(several~subkeys),~
7134   extra-left-margin,~
7135   extra-right-margin,~
7136   first-col,~
7137   first-row,~
7138   hlines,~
7139   hvlines,~
7140   l,~
7141   last-col,~
7142   last-row,~
7143   left-margin,~
7144   light-syntax,~
7145   name,~
7146   nullify-dots,~
7147   r,~
7148   renew-dots,~
7149   right-margin,~
7150   rules~(with~the~subkeys~'color'~and~'width'),~
7151   small,~
7152   t,~
7153   vl原因,~
7154   xdots/color,~
7155   xdots/shorten~and~
7156   xdots/line-style.
7157 }
7158 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7159 {
7160   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~

```

```

7161 \{NiceTabular\}. \\
7162 If-you-go-on,~it~will~be~ignored. \\
7163 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7164 }
7165 {
7166 The~available~keys~are~(in~alphabetic~order):~
7167 b,~
7168 baseline,~
7169 c,~
7170 cell-space-bottom-limit,~
7171 cell-space-limits,~
7172 cell-space-top-limit,~
7173 code-after,~
7174 code-for-first-col,~
7175 code-for-first-row,~
7176 code-for-last-col,~
7177 code-for-last-row,~
7178 colortbl-like,~
7179 columns-width,~
7180 corners,~
7181 create-extra-nodes,~
7182 create-medium-nodes,~
7183 create-large-nodes,~
7184 extra-left-margin,~
7185 extra-right-margin,~
7186 first-col,~
7187 first-row,~
7188 hlines,~
7189 hvlines,~
7190 last-col,~
7191 last-row,~
7192 left-margin,~
7193 light-syntax,~
7194 name,~
7195 notes/bottomrule,~
7196 notes/para,~
7197 nullify-dots,~
7198 renew-dots,~
7199 right-margin,~
7200 rules~(with~the~subkeys~'color'~and~'width'),~
7201 t,~
7202 tabularnote,~
7203 vlines,~
7204 xdots/color,~
7205 xdots/shorten~and~
7206 xdots/line-style.
7207 }
7208 \@@_msg_new:nnn { Duplicate-name }
7209 {
7210 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
7211 the~same~environment~name~twice.~You~can~go~on,~but,~
7212 maybe,~you~will~have~incorrect~results~especially~
7213 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
7214 message~again,~use~the~key~'allow-duplicate-names'~in~
7215 '\token_to_str:N \NiceMatrixOptions'.\\
7216 For~a~list~of~the~names~already~used,~type~H~<return>. \\
7217 }
7218 {
7219 The~names~already~defined~in~this~document~are:~
7220 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7221 }
7222 \@@_msg_new:nn { Option-auto-for-columns-width }
7223 {

```

```

7224   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
7225   If~you~go~on,~the~key~will~be~ignored.
7226 }

```

19 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.

Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).

Options are now available locally in `{pNiceMatrix}` and its variants.

The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.

New option `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 `{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 `code-for-first-row` is now available in the command `\NiceMatrixOptions`.

Following a discussion on TeX StackExchange⁶⁵, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁶⁶

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`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

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).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `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 `\dashline` 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`.

⁶⁵cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁶⁶Before 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 `overlay` and `remember picture`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.
Error message when the user gives an incorrect value for `last-row`.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).
The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.
Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.
The option `columns-width=auto` doesn’t need any more a second compilation.
The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).
The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁶⁷, 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`.

⁶⁷cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

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

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user 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 `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

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

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

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

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It's possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\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 `{NiceArray}`.

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

Changes between versions 3.15 and 4.0

New environment `{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 `\diagbox`

The key `hvhline` don't draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It's now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvhlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

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

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `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 `hvlines` 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).

Changes between versions 5.13 and 5.14

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 (that key will probably be deleted in a future version of `nicematrix`).

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`.

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	
@@ commands:	
<code>\@@_Block:</code>	1240, 5258
<code>\@@_Block_i</code>	5263, 5264, 5268
<code>\@@_Block_ii:nnnnn</code>	5268, 5269
<code>\@@_Block_iv:nnnnn</code>	5305, 5309
<code>\@@_Block_iv:nnnnnn</code>	5504, 5506
<code>\@@_Block_v:nnnnn</code>	5306, 5411
<code>\@@_Block_v:nnnnnn</code>	5533, 5536
<code>\@@_Cdots</code>	1163, 1230, 3809
<code>\g_@@_Cdots_lines_tl</code>	1258, 2951
<code>\@@_Cell:</code>	199, 218, 882, 1790, 1840, 1888, 2033, 2108, 2129, 2150, 2763
<code>\@@_CodeAfter:</code>	1244, 6102
<code>\@@_CodeAfter_i:n</code>	885, 2641, 2686, 6102, 6103, 6113
<code>\@@_CodeAfter_ii:n</code>	6106, 6108
<code>\@@_CodeAfter_keys:</code>	2891, 2912
<code>\@@_CodeBefore:w</code>	1432, 1434
<code>\@@_CodeBefore_keys:</code>	1412, 1429
<code>\@@_Ddots</code>	1165, 1232, 3841
<code>\g_@@_Ddots_lines_tl</code>	1261, 2949
<code>\g_@@_HVdotsfor_lines_tl</code>	1263, 2947, 3925, 4002, 6661
<code>\@@_Hdotsfor:</code>	1168, 1237, 3901
<code>\@@_Hdotsfor:nnnn</code>	3927, 3940
<code>\@@_Hdotsfor_i</code>	3910, 3916, 3923
<code>\@@_Hline:</code>	1235, 4752
<code>\@@_Hline_i:n</code>	4752, 4753, 4759
<code>\@@_Hline_ii:nn</code>	4756, 4759
<code>\@@_Hline_iii:n</code>	4757, 4760
<code>\@@_Hspace:</code>	1236, 3895
<code>\@@_Iddots</code>	1166, 1233, 3865
<code>\g_@@_Iddots_lines_tl</code>	1262, 2950
<code>\@@_Ldots</code>	1162, 1167, 1229, 3793
<code>\g_@@_Ldots_lines_tl</code>	1259, 2952
<code>\l_@@_Matrix_bool</code>	279, 1637, 1661, 2247, 2585, 2589, 2597, 2754
<code>\l_@@_NiceArray_bool</code>	276, 408, 1304, 1518, 1577, 1700, 1707, 1719, 2247, 2573, 2585, 2589, 2597, 2730, 4608, 4612, 4738, 4742, 4959, 4966, 6020
<code>\g_@@_NiceMatrixBlock_int</code>	265, 5018, 5023, 5026, 5037
<code>\l_@@_NiceTabular_bool</code>	169, 277, 890, 1089, 1411, 1414, 1485, 1610, 1614, 1708, 1720, 2574, 2783, 2792, 2890, 2893, 5338, 5420, 6028
<code>\@@_NotEmpty:</code>	1246, 2777
<code>\@@_OnlyMainNiceMatrix:n</code>	1242, 4465
<code>\@@_OnlyMainNiceMatrix_i:n</code>	4468, 4475, 4478
<code>\@@_RowStyle:n</code>	1247, 1356
<code>\@@_SubMatrix</code>	2872, 6252
<code>\@@_SubMatrix_in_code_before</code> ...	1404, 6232
<code>\@@_Vdots</code>	1164, 1231, 3825
<code>\g_@@_Vdots_lines_tl</code>	1260, 2948
<code>\@@_Vdotsfor:</code>	1238, 4000
<code>\@@_Vdotsfor:nnnn</code>	4004, 4015
<code>\@@_W:</code>	1664, 1747, 2095
<code>\l_@@_X_columns_aux_bool</code>	307, 1537, 1548, 2027
<code>\l_@@_X_columns_dim</code> ...	308, 1538, 1549, 2028
<code>\@@_actually_color:</code>	1413, 4140
<code>\@@_actually_diagbox:nnnnnn</code>	5316, 5602, 6040, 6056
<code>\@@_actually_draw_Cdots:</code>	3365, 3369
<code>\@@_actually_draw_Ddots:</code>	3515, 3519
<code>\@@_actually_draw_Iddots:</code>	3567, 3571
<code>\@@_actually_draw_Ldots:</code> ..	3326, 3330, 3991
<code>\@@_actually_draw_Vdots:</code> ..	3447, 3451, 4066
<code>\@@_adapt_S_column:</code>	230, 235, 248, 1484
<code>\@@_add_to_colors_seq:nn</code>	4127, 4139, 4162, 4177, 4192, 4201
<code>\@@_adjust_pos_of_blocks_seq:</code> ..	2859, 2914
<code>\@@_adjust_pos_of_blocks_seq_i:nnnn</code> ..	2917, 2919
<code>\@@_adjust_size_box:</code>	963, 990, 1897, 2159, 2665, 2710
<code>\@@_adjust_to_submatrix:nn</code>	3210, 3313, 3352, 3433, 3508, 3560
<code>\@@_adjust_to_submatrix:nnnnnn</code> ..	3217, 3219
<code>\@@_after_array:</code>	1646, 2796
<code>\g_@@_after_col_zero_bool</code>	309, 1130, 2642, 3907
<code>\@@_analyze_end:Nn</code>	2395, 2440
<code>\l_@@_argspec_tl</code>	3791, 3792, 3793, 3809, 3825, 3841, 3865, 3921, 3922, 3923, 3998, 3999, 4000, 4076, 4077, 4078, 6250, 6251, 6252
<code>\@@_array:</code>	1084, 2396, 2423
<code>\@@_arraycolor</code>	1401, 4238
<code>\c_@@_arydshln_loaded_bool</code> ...	24, 31, 1776
<code>\l_@@_auto_columns_width_bool</code>	511, 658, 2507, 2511, 5013
<code>\g_@@_aux_found_bool</code>	1275, 1280, 1436, 1506, 1509
<code>\g_@@_aux_tl</code>	281, 1512, 1546, 1653, 2805, 2819, 2827, 2900, 4835
<code>\l_@@_bar_at_end_of_pream_bool</code>	1701, 1834, 2577
<code>\l_@@_baseline_tl</code>	502, 503, 651, 652, 653, 654, 1097, 1579, 2200, 2212, 2217, 2219, 2224, 2229, 2319, 2320, 2324, 2329, 2331, 2336
<code>\@@_begin_of_NiceMatrix:nn</code>	2752, 2773
<code>\@@_begin_of_row:</code>	888, 912, 1189, 2643

\l_@@_block_auto_columns_width_bool ..	\l_@@_colortbl_like_bool 488, 674, 1248, 1689
..... 1498, 2512, 5006, 5011, 5021, 5031	\c_@@_colortbl_loaded_bool . 100, 105, 1197
\g_@@_block_box_int 352, 1478,	\l_@@_cols_tl 4170, 4185, 4215, 4249, 4257, 4258, 4353, 4356
5311, 5325, 5327, 5364, 5376, 5388, 5397, 5407	\g_@@_cols_vlism_seq .. 289, 1684, 1767, 6351
\@@_block_tikz:nnnnn 5590, 5930	\@@_columncolor 1402, 4173
\g_@@_blocks_dp_dim 273, 971, 974, 975, 5391, 5394	\@@_columncolor:n 4179, 4182
\g_@@_blocks_ht_dim 272, 977, 980, 981, 5382, 5385	\@@_columncolor_preamble 1174, 4453
\g_@@_blocks_seq 324, 1500, 2257, 5401, 5413, 5504	\c_@@_columncolor_regex 59, 1692
\g_@@_blocks_wd_dim 271, 965, 968, 969, 5370, 5373	\l_@@_columns_width_dim 266, 659, 801, 2508, 2514, 5019, 5025
\c_@@_booktabs_loaded_bool 25, 34, 1178, 2290	\g_@@_com_or_env_str 293, 296
\l_@@_borders_clist 341, 5474,	\@@_computations_for_large_nodes: ... 5135, 5148, 5153
5563, 5872, 5888, 5890, 5892, 5894, 5913, 5925	\@@_computations_for_medium_nodes: .. 5055, 5124, 5134, 5145
\@@_cartesian_path: 4171, 4186, 4300, 4312, 4405	\@@_compute_a_corner:nnnnnn 4823, 4825, 4827, 4829, 4842
\@@_cartesian_path:n 4216, 4347, 4405	\@@_compute_corners: 2858, 4815
\l_@@_cell_box 889, 937, 939, 945,	\@@_construct_preamble: 1301, 1658
951, 954, 958, 967, 968, 973, 974, 979, 980,	\l_@@_corners_cells_seq 328, 2864, 4350, 4390, 4539, 4545, 4552,
991, 992, 993, 994, 996, 999, 1003, 1005,	4674, 4680, 4687, 4817, 4833, 4837, 4838, 4898
1024, 1039, 1041, 1048, 1049, 1062, 1180,	\l_@@_corners_clist 507, 636, 641, 4505, 4641, 4818
1288, 1290, 1866, 1870, 1874, 1877, 1887,	\@@_create_col_nodes: 2399, 2427, 2446
1898, 2036, 2149, 2160, 2644, 2668, 2671,	\@@_create_diag_nodes: ... 1385, 2835, 3018
2673, 2690, 2713, 2717, 5610, 5720, 5757, 6035	\@@_create_extra_nodes: 1467, 2256, 3010, 5045
\l_@@_cell_space_bottom_limit_dim ... 491, 564, 994, 1345	\@@_create_large_nodes: 5053, 5129
\l_@@_cell_space_top_limit_dim 490, 562, 992, 1336	\@@_create_medium_and_large_nodes: .. 5050, 5140
\l_@@_cell_type_tl 269,	\@@_create_medium_nodes: 5051, 5119
270, 1790, 1889, 2034, 2108, 2151, 5283, 5285	\@@_create_nodes: 5126, 5137, 5147, 5150, 5191
\@@_cellcolor 1396, 4218, 4230, 4231	\@@_create_row_node: 1100, 1133, 1179
\@@_cellcolor_tabular 1172, 4430	\@@_cut_on_hyphen:w 4153, 4208, 4213, 4273, 4360,
\g_@@_cells_seq 2434, 2435, 2436, 2438	4361, 4383, 4384, 4414, 4415, 5790, 5799,
\@@_center_cell_box: 1854, 1861	5830, 5833, 5877, 5880, 6236, 6241, 6266, 6269
\@@_chessboardcolors 1403, 4223	\g_@@_ddots_int 2838, 3539, 3540
\@@_cline 148, 1228	\@@_def_env:nnn 2736, 2747, 2748, 2749, 2750, 2751
\@@_cline_i:nn 149, 150, 162, 165	\@@_define_L_C_R: 253, 1300
\@@_cline_i:w 150, 151	\c_@@_define_L_C_R_bool ... 252, 1300, 6575
\l_@@_code_before_bool 313,	\@@_define_com:nnn 6004, 6012, 6013, 6014, 6015, 6016
648, 675, 1104, 1294, 1326, 1515, 2454,	\@@_delimiter:nnn 1931, 1952, 1960, 1973, 1979, 1985, 6116
2471, 2489, 2520, 2546, 2580, 2617, 2905, 2907	\l_@@_delimiters_color_tl 524, 734, 1425, 1602, 1603, 1620, 1621,
\g_@@_code_before_tl . 1504, 1513, 1516, 2902	6096, 6154, 6155, 6182, 6501, 6502, 6525, 6526
\l_@@_code_before_tl 312, 647, 1325, 1412, 1516	\l_@@_delimiters_max_width_bool 525, 732, 1625
\l_@@_code_for_first_col_tl 581, 2655	\g_@@_delta_x_one_dim 2840, 3542, 3552
\l_@@_code_for_first_row_tl . 585, 900, 5689	\g_@@_delta_x_two_dim 2842, 3590, 3600
\l_@@_code_for_last_col_tl 583, 2699	\g_@@_delta_y_one_dim 2841, 3544, 3552
\l_@@_code_for_last_row_tl . 587, 907, 5692	\g_@@_delta_y_two_dim 2843, 3592, 3600
\l_@@_code_tl 301, 6227, 6449	\@@_diagbox:nn 1245, 6036
\l_@@_col_max_int 335, 3077, 3088, 3156, 3215, 3232	\@@_dotfill: 6025
\l_@@_col_min_int 334, 3082, 3145, 3150, 3213, 3230	\@@_dotfill_i: 6030, 6032
\g_@@_col_total_int 268, 1007, 1254, 1368,	\@@_dotfill_ii: 6029, 6032, 6033
1379, 1449, 1563, 2076, 2077, 2539, 2540,	\@@_dotfill_iii: 6033, 6034
2620, 2624, 2629, 2630, 2689, 2800, 2802,	\@@_double_int_eval:n 4072, 4086, 4087
2814, 2976, 3400, 3418, 3478, 4061, 4455,	
5064, 5074, 5108, 5195, 5516, 5727, 6274, 6323	
\l_@@_color_tl 343, 5255, 5329, 5331	
\g_@@_colors_seq 1408, 4130, 4134, 4135, 4144	
\@@_colortbl_like: 1170, 1248	

\g_@@_dp_ante_last_row_dim	916, 1213	4237, 4252, 5464, 5499, 5870, 6100, 6212,
\g_@@_dp_last_row_dim		6223, 6230, 6275, 6574, 6584, 6589, 6616, 6626
.	916, 917, 1216, 1217, 1289, 1290, 1596	\@@_error:nn
\g_@@_dp_row_zero_dim		13,
.	936, 937, 1207, 1208, 1589, 2313, 2352	666, 1756, 1757, 1758, 1934, 1980, 1986,
\@@_draw_Cdots:nnn	3350	2097, 2098, 2099, 3796, 3799, 3812, 3815,
\@@_draw_Ddots:nnn	3506	3828, 3831, 3845, 3846, 3851, 3852, 3869,
\@@_draw_Iddots:nnn	3558	3870, 3875, 3876, 4831, 5875, 6217, 6305, 6308
\@@_draw_Ldots:nnn	3311	\@@_error:nnn
\@@_draw_Vdots:nnn	3431	14, 4103, 6380, 6415
\@@_draw_blocks:	2257, 5501	\@@_error_too_much_cols:
\@@_draw_dotted_lines:	2857, 2936	1729, 6646
\@@_draw_dotted_lines_i:	2939, 2943	\@@_everycr:
\l_@@_draw_first_bool	350, 3856, 3880, 3891	1126, 1202, 1205
\@@_draw_hlines:	2870, 4734	\@@_everycr_i:
\@@_draw_line:	3348,	1126, 1127
.	3393, 3504, 3556, 3604, 3606, 4125, 4974, 5004	\l_@@_except_borders_bool
\@@_draw_line_ii:nn	4105, 4109
\@@_draw_line_iii:nn	4112, 4116
\@@_draw_non_standard_dotted_line: . .		516, 609, 4608, 4612, 4738, 4742
.	3612, 3614	\@@_exec_code_before:
\@@_draw_non_standard_dotted_line:n . .		1294, 1406
.	3617, 3620	\@@_expand_clist:NN
\@@_draw_non_standard_dotted_line:nnn		4353, 4354, 4406
.	3622, 3627, 3641	\l_@@_exterior_arraycolsep_bool
\@@_draw_standard_dotted_line:	3611, 3642
\@@_draw_standard_dotted_line_i:	3705, 3709	504, 797, 1710, 1722, 2576
\l_@@_draw_tl	339, 5468,	\l_@@_extra_left_margin_dim
.	5472, 5551, 5772, 5778, 5780, 5782, 5819, 5820
\@@_draw_vlines:	2871, 4604
\g_@@_empty_cell_bool	321, 998, 1008,	520, 626, 1533, 2721, 3481
.	2678, 2725, 3807, 3823, 3839, 3863, 3886, 3897	\@@_extract_brackets
\@@_end_Cell:	204, 220, 984,	5575, 5762
.	1792, 1855, 1893, 2039, 2110, 2139, 2155, 2763	\@@_extract_coords_values:
\l_@@_end_of_row_tl		5216, 5223
.	521, 522, 575, 2419, 2420, 6765	\@@_fatal:n
\c_@@_endpgfortikzpicture_tl		15, 285, 1488, 1912, 1941,
.	43, 47, 2940, 4113, 4945	2404, 2408, 2410, 2443, 3912, 6651, 6654, 6657
\c_@@_enumitem_loaded_bool		\@@_fatal:nn
.	26, 37, 381, 702, 707, 718, 723	16, 1781, 2102
\@@_env:	260, 264, 922, 928, 1025,	\l_@@_fill_tl
.	1031, 1045, 1053, 1109, 1115, 1121, 1192,	338, 5466, 5573, 5575
.	1369, 1370, 1375, 1376, 1381, 1382, 1393,	\l_@@_final_i_int
.	1446, 1447, 1452, 1455, 1458, 1461, 2455,
.	2458, 2460, 2476, 2482, 2485, 2494, 2500,	2847, 3064, 3069, 3072, 3097,
.	2503, 2525, 2531, 2534, 2551, 2557, 2563,	3105, 3109, 3118, 3126, 3206, 3261, 3342,
.	2587, 2595, 2609, 2620, 2624, 2630, 2881,	3415, 3421, 3424, 3945, 3973, 4041, 4051, 4053
.	2978, 2982, 2988, 2995, 3001, 3035, 3037,	\l_@@_final_j_int
.	3044, 3046, 3047, 3052, 3055, 3117, 3185,
.	3249, 3260, 3273, 3276, 3295, 3298, 3403,	2848, 3065, 3070, 3077, 3082, 3088, 3098,
.	3406, 3421, 3424, 3954, 3972, 4029, 4047,	3106, 3110, 3119, 3127, 3207, 3262, 3295,
.	4098, 4100, 4119, 4122, 4853, 4872, 4890,	3298, 3306, 3532, 3966, 3976, 3978, 4020, 4049
.	5077, 5079, 5087, 5198, 5207, 5225, 5625,	\l_@@_final_open_bool
.	5634, 5638, 5652, 5657, 5669, 5675, 5676,
.	5679, 5697, 5734, 6131, 6134, 6290, 6292,	2850, 3071,
.	6297, 6299, 6326, 6328, 6333, 6335, 6433, 6445	3075, 3078, 3085, 3091, 3095, 3111, 3339,
\g_@@_env_int		3374, 3379, 3390, 3454, 3464, 3469, 3490,
.	259, 260, 262, 1497, 1507, 1510, 1652, 5234	3529, 3579, 3713, 3728, 3759, 3760, 3943,
\@@_error:n	12,	3967, 3979, 4018, 4042, 4054, 4095, 4942, 4979
.	384, 409, 534, 544, 597, 728, 784, 794, 800,	\@@_find_extremities_of_line:nnnn . . .
.	809, 817, 835, 842, 850, 851, 852, 858,
.	863, 864, 865, 876, 878, 879, 880, 1354,	3059, 3316, 3355, 3436, 3511, 3563
.	1427, 1558, 1568, 1640, 2234, 2295, 2341,	\l_@@_first_col_int
	
		136, 149, 362, 363, 577, 856, 888,
		1379, 1449, 1572, 1702, 2449, 2469, 2812,
		2976, 3400, 3418, 3905, 4374, 4467, 5064,
		5074, 5108, 5156, 5195, 5985, 5991, 5997, 6323
		\l_@@_first_i_tl
		6262, 6267, 6286, 6317,
		6326, 6328, 6383, 6390, 6392, 6474, 6485, 6489
		\l_@@_first_j_tl
		6263, 6268, 6290,
		6292, 6353, 6366, 6373, 6375, 6475, 6486, 6490
		\l_@@_first_row_int
	
		360, 361, 578, 860, 1252,
		1373, 1444, 1587, 2231, 2310, 2338, 2349,
		2809, 2974, 3270, 3292, 5057, 5071, 5098,
		5155, 5193, 5631, 5649, 5983, 6128, 6287, 6783
		\c_@@_footnote_bool
		1473, 1656, 6544, 6587, 6610, 6613, 6623, 6629
		\c_@@_footnotehyper_bool
		6543, 6588, 6620
		\@@_full_name_env:
	
		294, 6635, 6667, 6674, 6682, 6690, 6694,
		6797, 6810, 6813, 6826, 6831, 6836, 6838,
		6868, 6887, 6892, 6897, 6904, 6930, 6939, 7110

```

\@@_hdottedline: ..... 1234, 4927
\@@_hdottedline:n ..... 4935, 4939
\@@_hdottedline_i: ..... 4930, 4932
\@@_hdottedline_i:n ..... 4944, 4948
\@@_hline:nn ..... 4622, 4749, 4768
\@@_hline_i:nn ..... 2868, 4625, 4628
\@@_hline_i_complete:nn ..... 2868, 4732
\@@_hline_ii:nnnn ... 4650, 4661, 4694, 4733
\l_@@_hlines_clist ..... 356, 589, 603,
608, 638, 1134, 1136, 1140, 2870, 4747, 4748
\l_@@_hpos_of_block_cap_bool .....
.... 347, 5487, 5490, 5493, 5628, 5698, 5735
\l_@@_hpos_of_block_tl ..... 345, 346,
5239, 5241, 5243, 5245, 5247, 5249, 5284,
5285, 5287, 5337, 5343, 5353, 5404, 5427,
5431, 5443, 5448, 5480, 5482, 5484, 5486,
5489, 5492, 5701, 5713, 5724, 5728, 5738, 5750
\g_@@_ht_last_row_dim .....
..... 918, 1214, 1215, 1287, 1288, 1595
\g_@@_ht_row_one_dim .. 944, 945, 1211, 1212
\g_@@_ht_row_zero_dim .....
.... 938, 939, 1209, 1210, 1590, 2312, 2351
\@@_hvlines_block:nnn ..... 5545, 5826
\l_@@_hvlines_block_bool .. 351, 5476, 5541
\@@_i: ..... 5057, 5059,
5060, 5061, 5062, 5071, 5077, 5079, 5080,
5081, 5082, 5087, 5088, 5089, 5090, 5098,
5101, 5103, 5104, 5105, 5157, 5159, 5162,
5163, 5167, 5168, 5193, 5198, 5200, 5202,
5206, 5207, 5218, 5225, 5227, 5229, 5233, 5234
\g_@@_iddots_int ..... 2839, 3587, 3588
\l_@@_in_env_bool .... 275, 408, 1488, 1489
\c_@@_in_preamble_bool . 21, 22, 23, 698, 714
\l_@@_initial_i_int ..... 2845,
3062, 3137, 3140, 3165, 3173, 3177, 3186,
3194, 3204, 3250, 3335, 3381, 3383, 3397,
3403, 3406, 3944, 3945, 3955, 4023, 4033, 4035
\l_@@_initial_j_int .....
..... 2846, 3063, 3138, 3145,
3150, 3156, 3166, 3174, 3178, 3187, 3195,
3205, 3251, 3273, 3276, 3284, 3471, 3473,
3478, 3524, 3948, 3958, 3960, 4019, 4020, 4031
\l_@@_initial_open_bool .....
.... 2849, 3139, 3143, 3146, 3153, 3159,
3163, 3179, 3332, 3371, 3378, 3388, 3454,
3461, 3467, 3521, 3573, 3711, 3758, 3942,
3949, 3961, 4017, 4024, 4036, 4094, 4941, 4978
\@@_insert_tabularnotes: ..... 2262, 2265
\@@_instruction_of_type:nnn .....
..... 1065, 3801, 3817, 3833, 3856, 3880
\c_@@_integers_alist_tl ..... 6459, 6470
\l_@@_inter_dots_dim .....
. 492, 493, 2854, 3716, 3723, 3734, 3742,
3749, 3754, 3766, 3774, 4970, 4972, 5000, 5002
\g_@@_internal_code_after_tl 302, 1826,
1930, 1951, 1959, 1972, 1978, 1984, 1994,
2886, 2887, 4767, 4934, 5314, 5600, 6038, 6244
\@@_intersect_our_row:nnnn ..... 4336
\@@_intersect_our_row_p:nnnn ..... 4287
\@@_j: ..... 5064, 5066,
5067, 5068, 5069, 5074, 5077, 5079, 5082,
5084, 5085, 5087, 5090, 5092, 5093, 5108,
5111, 5113, 5114, 5115, 5170, 5172, 5175,
5177, 5181, 5182, 5195, 5198, 5199, 5201,
5206, 5207, 5219, 5225, 5226, 5228, 5233, 5234
\l_@@_l_dim .....
.... 3689, 3690, 3703, 3704, 3716, 3722,
3733, 3741, 3749, 3754, 3766, 3767, 3774, 3775
\l_@@_large_nodes_bool 515, 616, 5049, 5053
\g_@@_last_col_found_bool ..... 370,
1257, 1564, 1632, 2538, 2612, 2687, 2799, 5725
\l_@@_last_col_int .....
..... 368, 369, 785, 828, 830, 843, 859,
877, 1278, 1281, 1567, 1714, 2759, 2761,
2800, 2802, 3441, 3476, 3798, 3814, 3852,
3876, 5509, 5514, 5515, 5516, 5519, 5548,
5558, 5570, 5582, 5594, 5606, 5621, 5652,
5657, 5665, 5681, 5987, 5993, 5999, 6650, 6668
\l_@@_last_col_without_value_bool ...
..... 367, 827, 2801, 6653
\l_@@_last_empty_column_int .....
..... 4863, 4864, 4877, 4883, 4896
\l_@@_last_empty_row_int .....
..... 4845, 4846, 4859, 4880
\l_@@_last_i_tl ..... 6264,
6270, 6273, 6286, 6320, 6333, 6335, 6383, 6390
\l_@@_last_j_tl .....
6265, 6271, 6274, 6297, 6299, 6356, 6366, 6373
\l_@@_last_row_int .....
.... 364, 365, 579, 905, 952, 1146, 1272,
1276, 1283, 1552, 1556, 1559, 1571, 1593,
2421, 2422, 2651, 2652, 2696, 2697, 2804,
3321, 3360, 3830, 3846, 3870, 4473, 4481,
5508, 5511, 5512, 5531, 5548, 5558, 5570,
5582, 5593, 5605, 5619, 5680, 5691, 5995, 6929
\l_@@_last_row_without_value_bool ...
..... 366, 1274, 1554, 2803
\l_@@_left_delim_dim .....
..... 1302, 1306, 1311, 2387, 2674
\g_@@_left_delim_tl .....
1310, 1475, 1604, 1628, 1698, 1915, 1917, 4969
\l_@@_left_margin_dim .....
..... 517, 619, 1316, 2675, 4960, 5186
\l_@@_letter_for_dotted_lines_str ...
..... 808, 819, 820, 1762
\l_@@_letter_vlism_tl ..... 288, 596, 1765
\l_@@_light_syntax_bool 501, 573, 1319, 1528
\@@_light_syntax_i ..... 2412, 2415
\@@_line ..... 2885, 4078
\@@_line_i:nn ..... 4085, 4092
\l_@@_line_width_dim .....
344, 5478, 5773, 5811, 5822, 5828, 5840,
5867, 5887, 5902, 5904, 5914, 5915, 5917, 5928
\@@_line_with_light_syntax:n ... 2426, 2430
\@@_line_with_light_syntax:i:n .....
..... 2425, 2431, 2432
\@@_math_toggle_token: .....
168, 986, 2645, 2662, 2691, 2707, 6080, 6084
\g_@@_max_cell_width_dim .....
..... 995, 996, 1499, 2513, 5012, 5038
\c_@@_max_l_dim ..... 3703, 3708
\l_@@_medium_nodes_bool 514, 615, 5047, 5672
\@@_message_hdotsfor: 6659, 6667, 6674, 6682
\@@_msg_new:nn ..... 17, 6559,
6565, 6592, 6601, 6631, 6664, 6671, 6679,
6687, 6692, 6698, 6703, 6708, 6713, 6718,

```


6723, 6728, 6733, 6739, 6745, 6751, 6756, 6762, 6768, 6773, 6780, 6787, 6794, 6800, 6809, 6811, 6817, 6834, 6841, 6846, 6853, 6860, 6866, 6873, 6880, 6885, 6895, 6901, 6908, 6915, 6921, 6927, 6935, 6937, 6943, 7222	\@@_old_ldots 1221, 3806
\@@_msg_new:nnn ... 18, 6545, 6823, 6948, 6958, 6973, 6994, 7012, 7055, 7107, 7158, 7208	\@@_old_multicolumn 3900
\@@_msg_redirect_name:nn 19, 803, 1644, 5522, 5525	\@@_old_pgfpoinanchor 176, 6451, 6455
\@@_multicolumn:nnn 1239, 2045	\@@_old_pgful@check@rerun 93, 97
\g_@@_multicolumn_cells_seq 330, 1250, 1296, 2060, 2825, 2829, 2830, 5082, 5090, 5212, 5636, 5654	\@@_old_vdots 1223, 3838
\g_@@_multicolumn_sizes_seq 331, 1251, 1297, 2062, 2831, 2832, 5213	\@@_open_x_final_dim: 3289, 3341, 3375, 3533, 3582
\g_@@_name_env_str 292, 297, 298, 1482, 1483, 2442, 2731, 2732, 2740, 2741, 2770, 2781, 2789, 2908, 6008, 6648	\@@_open_x_initial_dim: 3267, 3334, 3372, 3526, 3576
\l_@@_name_str 513, 668, 924, 927, 1027, 1030, 1117, 1120, 2459, 2460, 2484, 2485, 2502, 2503, 2533, 2534, 2559, 2562, 2605, 2608, 2626, 2629, 3036, 3037, 3048, 3051, 3054, 5203, 5206, 5230, 5233	\@@_open_y_final_dim: 3413, 3465, 3531, 3581
\g_@@_names_seq 274, 665, 667, 7220	\@@_open_y_initial_dim: 3395, 3462, 3523, 3575
\l_@@_nb_cols_int 5961, 5972, 5980, 5986, 5992, 5998	\l_@@_parallelize_diags_bool 505, 506, 611, 2836, 3537, 3585
\g_@@_nb_of_X_int 306, 1535, 1543, 1683, 2026	\@@_patch_m_preamble:n 2054, 2080, 2117, 2122, 2188
\l_@@_nb_rows_int 5960, 5971, 5989	\@@_patch_m_preamble_i:n 2084, 2085, 2086, 2104
\@@_newcolumnntype 1153, 1663, 1664	\@@_patch_m_preamble_ii:nn 2087, 2088, 2089, 2114
\@@_node_for_cell: 1004, 1011, 1417, 2672, 2722	\@@_patch_m_preamble_iii:n 2090, 2119
\@@_node_for_multicolumn:nn 5214, 5221	\@@_patch_m_preamble_iv:nnn 2091, 2092, 2093, 2124
\@@_node_left:nn 6432, 6433, 6493	\@@_patch_m_preamble_ix:n 2173, 2191
\@@_node_position: 1375, 1377, 1381, 1383, 1446, 1448, 1456, 1462	\@@_patch_m_preamble_v:nnnn 2094, 2095, 2144
\@@_node_position_i: 1459, 1463	\@@_patch_m_preamble_vi:n 2096, 2165
\@@_node_right:nnnn 6442, 6444, 6517	\@@_patch_m_preamble_x:n 2112, 2142, 2163, 2168, 2170, 2194
\g_@@_not_empty_cell_bool 311, 1002, 1009, 2778	\@@_patch_node_for_cell: 1037, 1417
\@@_not_in_exterior:nnnn 4328	\@@_patch_node_for_cell:n 1035, 1061, 1064
\@@_not_in_exterior_p:nnnn 4265	\@@_patch_preamble:n 1686, 1732, 1771, 1779, 1800, 1831, 1919, 1935, 1937, 1953, 1961, 1987, 1996, 2016
\l_@@_notes_above_space_dim 508, 509	\@@_patch_preamble_i:n 1736, 1737, 1738, 1786
\l_@@_notes_bottomrule_bool 686, 846, 871, 2288	\@@_patch_preamble_ii:nn 1739, 1740, 1741, 1797
\l_@@_notes_code_after_tl 684, 2297	\@@_patch_preamble_iii:n . 1742, 1802, 1810
\l_@@_notes_code_before_tl 682, 2269	\@@_patch_preamble_iii_i:n 1805, 1807
\@@_notes_label_in_list:n 377, 396, 404, 694	\@@_patch_preamble_iv:nnn 1743, 1744, 1745, 1835, 2028
\@@_notes_label_in_tabular:n . 376, 417, 691	\@@_patch_preamble_ix:n 2001, 2019
\l_@@_notes_para_bool .. 680, 844, 869, 2273	\@@_patch_preamble_v:nnnn 1746, 1747, 1882
\@@_notes_style:n 375, 378, 396, 404, 420, 425, 688	\@@_patch_preamble_vi:n 1748, 1904
\l_@@_nullify_dots_bool 510, 614, 3805, 3821, 3837, 3861, 3884	\@@_patch_preamble_vii:nn 1749, 1750, 1751, 1910
\l_@@_number_of_notes_int 374, 411, 421, 431	\@@_patch_preamble_vii_i:nn 1923, 1926, 1928
\@@_old_CT@arc@ 1490, 2910	\@@_patch_preamble_viii:nn 1752, 1753, 1754, 1939
\@@_old_cdots 1222, 3822	\@@_patch_preamble_viii_i:nnn .. 1943, 1965
\@@_old_ddots 1224, 3862	\@@_patch_preamble_x:n 1795, 1859, 1902, 1908, 1998, 2022, 2042
\@@_old_dotfill 6024, 6027, 6035	\@@_patch_preamble_xi:n 1763, 1990
\@@_old_dotfill: 1243	\@@_patch_preamble_xii: 1755, 2024
\l_@@_old_iRow_int 303, 1268, 2956	\@@_pgf_rect_node:nnn 464, 1460, 5674
\@@_old_ialign: 1099, 1218, 5503	\@@_pgf_rect_node:nnnnn 439, 5197, 5224, 5624, 5668, 6419
\@@_old_iddots 1225, 3885	\c_@@_pgfortikzpicture_tl 42, 46, 2938, 4111, 4943
\l_@@_old_jCol_int 304, 1270, 2957	\@@_pgfpoinanchor:n 6447, 6452
	\@@_pgfpoinanchor_i:nn 6455, 6457
	\@@_pgfpoinanchor_ii:w 6458, 6466
	\@@_pgfpoinanchor_iii:w 6479, 6481

```

\@@_picture_position: .....
..... 1370, 1377, 1383, 1448, 1462, 1463
\g_@@_pos_of_blocks_seq .....
..... 325, 1295, 1501, 2063, 2817, 2821, 2822,
..... 2862, 2916, 4263, 4499, 4635, 4910, 5538, 6048
\g_@@_pos_of_stroken_blocks_seq .....
..... 327, 1502, 4503, 4639, 5560
\g_@@_pos_of_xdots_seq .....
..... 326, 1503, 2863, 3202, 4501, 4637
\g_@@_post_treatment_cell_tl .....
..... 884, 988, 1335, 1344, 1863, 2035
\@@_pre_array: ..... 1266, 1327, 1525
\@@_pre_array_i:w ..... 1323, 1525
\@@_pre_array_ii: ..... 1176, 1298
\@@_pre_code_before: ..... 1363, 1438
\c_@@_preamble_first_col_tl .... 1703, 2637
\c_@@_preamble_last_col_tl .... 1715, 2682
\g_@@_preamble_tl .....
..... 1477, 1665, 1669, 1673, 1679, 1694,
..... 1703, 1712, 1715, 1724, 1728, 1769, 1778,
..... 1788, 1799, 1812, 1837, 1884, 1906, 1922,
..... 1950, 1958, 1971, 1977, 1992, 2005, 2012,
..... 2021, 2030, 2053, 2055, 2106, 2116, 2121,
..... 2126, 2146, 2167, 2177, 2184, 2193, 2396, 2423
\@@_pred:n .....
..... 137, 167, 2761, 4540, 4553, 4675, 4688
\@@_provide_pgfsyspdfmark: ... 60, 69, 1472
\@@_put_box_in_flow: ..... 1630, 2196, 2389
\@@_put_box_in_flow_bis:nn .... 1627, 2356
\@@_put_box_in_flow_i: ..... 2202, 2204
\@@_qpoint:n ..... 263, 2207, 2209, 2221,
..... 2237, 2304, 2306, 2322, 2333, 2344, 3024,
..... 3026, 3028, 3030, 3040, 3042, 3284, 3306,
..... 3335, 3342, 3381, 3383, 3397, 3415, 3471,
..... 3473, 3524, 3532, 4119, 4122, 4373, 4377,
..... 4393, 4395, 4563, 4565, 4567, 4698, 4700,
..... 4702, 4951, 4955, 4962, 4996, 4999, 5001,
..... 5103, 5113, 5615, 5617, 5619, 5621, 5645,
..... 5665, 5694, 5795, 5797, 5804, 5806, 5841,
..... 5843, 5847, 5852, 5854, 5858, 5901, 5903,
..... 5905, 5912, 5916, 5918, 6061, 6063, 6066,
..... 6068, 6121, 6123, 6317, 6320, 6358, 6375, 6392
\l_@@_radius_dim ..... 496, 497, 1993,
..... 2853, 3346, 3347, 3783, 4929, 4953, 4997, 4998
\l_@@_real_left_delim_dim 2358, 2373, 2388
\l_@@_real_right_delim_dim 2359, 2385, 2391
\@@_recreate_cell_nodes: ..... 1386, 1442
\g_@@_recreate_cell_nodes_bool .....
..... 512, 1187, 1386, 1409, 1416, 1421
\@@_rectanglecolor .. 1397, 4188, 4221, 4241
\@@_rectanglecolor:nnn ... 4194, 4203, 4206
\@@_renew_NC@rewrite@S: 187, 191, 211, 1256
\@@_renew_dots: ..... 1160, 1249
\l_@@_renew_dots_bool .....
..... 612, 793, 1249, 6576, 6583
\@@_renew_matrix: 788, 792, 5940, 6578, 6582
\l_@@_respect_blocks_bool 4247, 4261, 4284
\@@_restore_iRow_jCol: ..... 2909, 2954
\@@_revtex_array: ..... 1076, 1087
\c_@@_revtex_bool ..... 50, 52, 55, 57, 1086
\l_@@_right_delim_dim .....
..... 1303, 1307, 1313, 2390, 2719
\g_@@_right_delim_tl .. 1312, 1476, 1622,
..... 1628, 1699, 1918, 1947, 1948, 1969, 1974, 4971
\l_@@_right_margin_dim .....
..... 518, 621, 1532, 2720, 3480, 4967, 5189
\@@_rotate: ..... 1241, 4071
\g_@@_rotate_bool .....
..... 280, 961, 989, 1896, 2158, 2664,
..... 2709, 4071, 5337, 5361, 5366, 5426, 5442, 5611
\@@_rotate_cell_box: .....
..... 949, 989, 1896, 2158, 2664, 2709, 5611
\l_@@_rounded_corners_dim .....
..... 342, 5470, 5583, 5787, 5788, 5823, 5869, 5926
\@@_roundedrectanglecolor ..... 1398, 4197
\l_@@_row_max_int .... 333, 3072, 3214, 3231
\l_@@_row_min_int .... 332, 3140, 3212, 3229
\g_@@_row_of_col_done_bool .....
..... 310, 1131, 1481, 2468
\g_@@_row_style_tl ..... 314,
..... 895, 914, 1333, 1342, 1358, 1505, 1847, 5332
\g_@@_row_total_int .... 267, 1253, 1367,
..... 1373, 1444, 1570, 2232, 2339, 2804, 2811,
..... 2974, 3270, 3292, 3986, 5057, 5071, 5098,
..... 5193, 5531, 5631, 5649, 6128, 6273, 6287, 6784
\@@_rowcolor ..... 1399, 4158
\@@_rowcolor:n ..... 4164, 4167
\@@_rowcolor_tabular ..... 1173, 4441
\@@_rowcolors ..... 1400, 4254
\@@_rowcolors_i:nnnn ..... 4288, 4323
\l_@@_rowcolors_restart_bool ... 4250, 4276
\g_@@_rows_seq . 2418, 2420, 2422, 2424, 2426
\l_@@_rows_tl .....
..... 4169, 4184, 4214, 4290, 4354, 4379
\l_@@_rules_color_tl .....
..... 305, 548, 1522, 1523, 6348, 6349
\@@_set_CT@arc@: ..... 170, 1523, 6349
\@@_set_CT@arc@_i: ..... 171, 172
\@@_set_CT@arc@_ii: ..... 171, 174
\@@_set_final_coords: ..... 3240, 3265
\@@_set_final_coords_from_anchor:n ..
..... 3256, 3345, 3376, 3457, 3466, 3536, 3584
\@@_set_initial_coords: ..... 3235, 3254
\@@_set_initial_coords_from_anchor:n .
..... 3245, 3338, 3373, 3456, 3463, 3528, 3578
\@@_set_size:n ..... 5958, 5973
\c_@@_siunitx_loaded_bool 177, 181, 186, 232
\c_@@_size_seq .....
..... 1276, 1281, 1365, 1366, 1367, 1368, 2807
\l_@@_small_bool ..... 786, 833, 839,
..... 861, 893, 1182, 1912, 1941, 2646, 2692, 2851
\@@_standard_cline ..... 133, 1227
\@@_standard_cline:w ..... 133, 134
\l_@@_standard_cline_bool .. 489, 560, 1226
\c_@@_standard_tl 499, 500, 3610, 4973, 5003
\g_@@_static_num_of_col_int .....
..... 337, 1639, 1687, 5519, 6683, 6695, 6888
\l_@@_stop_loop_bool ..... 3066, 3067,
..... 3099, 3112, 3121, 3134, 3135, 3167, 3180, 3189
\@@_store_in_tmpb_tl ..... 5765, 5767
\@@_stroke_block:nnn ..... 5555, 5769
\@@_stroke_borders_block:nnn ... 5567, 5865
\@@_stroke_horizontal:n .. 5893, 5895, 5910
\@@_stroke_vertical:n .... 5889, 5891, 5899
\@@_sub_matrix:nnnnnnn ..... 6255, 6259

```


<code>\@@_sub_matrix_i:nnnn</code>	6309, 6315	<code>\@@_use_arraybox_with_notes_b:</code> ..	1581, 2301
<code>\l_@@_submatrix_extra_height_dim</code>	353, 6174, 6343	<code>\@@_use_arraybox_with_notes_c:</code>	1582, 1613, 2245, 2315, 2354
<code>\l_@@_submatrix_hlines_clist</code>	358, 6186, 6204, 6382, 6384	<code>\@@_vdottedline:n</code>	1995, 4976
<code>\l_@@_submatrix_left_xshift_dim</code>	354, 6176, 6395, 6428	<code>\@@_vdottedline_i:n</code>	4983, 4988, 4992
<code>\l_@@_submatrix_name_str</code>	6219, 6277, 6417, 6419, 6431, 6433, 6441, 6445	<code>\@@_vline:nn</code>	1827, 4483, 4619
<code>\g_@@_submatrix_names_seq</code>	329, 2889, 6216, 6220, 6832	<code>\@@_vline_i:nn</code>	2867, 4488, 4492
<code>\l_@@_submatrix_right_xshift_dim</code>	355, 6178, 6404, 6438	<code>\@@_vline_i_complete:nn</code>	2867, 4602
<code>\g_@@_submatrix_seq</code> ...	336, 1293, 3216, 6242	<code>\@@_vline_ii:nnnn</code> ...	4515, 4526, 4559, 4603
<code>\l_@@_submatrix_slim_bool</code>	6184, 6285, 6805	<code>\l_@@_vlines_clist</code>	357, 590, 602, 607, 637, 1671, 1677, 1709, 1721, 2003, 2010, 2175, 2182, 2575, 2871, 4617, 4618
<code>\l_@@_submatrix_vlines_clist</code>	359, 6188, 6206, 6365, 6367	<code>\l_@@_vpos_of_block_tl</code>	348, 349, 5251, 5253, 5342, 5352, 5430, 5447, 5495, 5497
<code>\@@_succ:n</code>	162, 166, 1109, 1115, 1120, 1121, 1141, 1827, 1931, 2010, 2061, 2182, 2209, 2551, 2557, 2562, 2563, 2587, 2595, 2608, 2609, 2620, 2624, 2629, 2630, 3306, 3383, 3473, 3532, 4332, 4377, 4393, 4536, 4567, 4613, 4671, 4702, 4743, 4768, 4915, 4917, 4919, 4921, 4962, 5001, 5163, 5167, 5177, 5181, 5619, 5621, 5665, 5804, 5806, 5936, 6066, 6068, 6123	<code>\@@_w:</code>	1663, 1746, 2094
<code>\l_@@_suffix_tl</code>	5125, 5136, 5146, 5149, 5198, 5206, 5207, 5225, 5233, 5234	<code>\g_@@_width_first_col_dim</code>	323, 1480, 1575, 2463, 2667, 2668
<code>\c_@@_table_collect_begin_tl</code> ..	218, 243, 245	<code>\g_@@_width_last_col_dim</code>	322, 1479, 1634, 2616, 2712, 2713
<code>\c_@@_table_print_tl</code>	220, 246, 247	<code>\@@_write_aux_for_cell_nodes:</code> ..	2907, 2969
<code>\l_@@_tabular_width_dim</code>	278, 1092, 1094, 1726, 2790	<code>\l_@@_x_final_dim</code>	317, 3242, 3291, 3300, 3301, 3304, 3307, 3308, 3459, 3475, 3483, 3487, 3491, 3493, 3498, 3500, 3534, 3543, 3551, 3591, 3599, 3638, 3653, 3662, 3696, 3748, 3764, 4123, 4963, 4972, 4998, 6284, 6300, 6301, 6307, 6404, 6421, 6438
<code>\l_@@_tabularnote_tl</code>	373, 848, 873, 2261, 2270	<code>\l_@@_x_initial_dim</code>	315, 3237, 3269, 3278, 3279, 3282, 3285, 3286, 3459, 3474, 3475, 3482, 3487, 3491, 3493, 3495, 3498, 3500, 3525, 3543, 3551, 3591, 3599, 3635, 3652, 3662, 3696, 3748, 3762, 3764, 3782, 3784, 4120, 4956, 4970, 4997, 6283, 6293, 6294, 6304, 6395, 6420, 6428
<code>\g_@@_tabularnotes_seq</code>	372, 412, 2276, 2282, 2298	<code>\l_@@_xdots_color_tl</code>	523, 537, 3325, 3364, 3445, 3446, 3514, 3566, 3618, 3990, 4065, 4082
<code>\@@_test_hline_in_block:nnnn</code>	4636, 4638, 4771	<code>\l_@@_xdots_down_tl</code> ...	541, 3625, 3646, 3681
<code>\@@_test_hline_in_stroken_block:nnnn</code> ..	4640, 4793	<code>\l_@@_xdots_line_style_tl</code>	498, 500, 533, 3610, 3618, 4973, 5003
<code>\@@_test_if_cell_in_a_block:nn</code>	4849, 4867, 4885, 4905	<code>\l_@@_xdots_shorten_dim</code>	494, 495, 539, 2855, 3632, 3633, 3722, 3733, 3741
<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...	4911, 4913	<code>\l_@@_xdots_up_tl</code> ...	542, 3624, 3645, 3671
<code>\@@_test_if_math_mode:</code>	282, 1487, 2742	<code>\l_@@_y_final_dim</code>	318, 3243, 3343, 3347, 3385, 3389, 3391, 3416, 3426, 3427, 3545, 3548, 3593, 3596, 3638, 3653, 3661, 3698, 3753, 3772, 4124, 4954, 5002, 6124, 6146, 6161, 6321, 6336, 6337, 6342, 6360, 6377, 6421, 6429, 6439
<code>\@@_test_in_corner_h:</code>	4641, 4669	<code>\l_@@_y_initial_dim</code>	316, 3238, 3336, 3346, 3384, 3385, 3389, 3391, 3398, 3408, 3409, 3545, 3550, 3593, 3598, 3635, 3652, 3661, 3698, 3753, 3770, 3772, 3782, 3785, 4121, 4952, 4953, 4954, 5000, 6122, 6146, 6161, 6318, 6329, 6330, 6342, 6359, 6376, 6420, 6429, 6439
<code>\@@_test_in_corner_v:</code>	4506, 4534	<code>\@</code>	2409, 2431, 5987, 5993, 5999, 6547, 6548, 6568, 6598, 6607, 6635, 6695, 6700, 6705, 6710, 6715, 6759, 6764, 6770, 6777, 6784, 6790, 6791, 6806, 6814, 6820, 6826, 6827, 6838, 6850, 6857, 6869, 6876, 6883, 6892, 6898, 6905, 6912, 6918, 6924, 6936, 6940, 6945, 6951, 6960, 6961, 6975, 6976, 6992, 6996, 6997, 7015, 7016, 7058, 7059, 7110, 7111, 7161, 7162, 7215, 7216
<code>\@@_test_vline_in_block:nnnn</code>	4500, 4502, 4782		
<code>\@@_test_vline_in_stroken_block:nnnn</code> ..	4504, 4804		
<code>\l_@@_the_array_box</code>	1299, 1315, 1542, 2249, 2250, 2252, 2255		
<code>\c_@@_tikz_loaded_bool</code>	27, 41, 1388, 2873, 4980, 5462		
<code>\l_@@_tikz_seq</code>	340, 5463, 5586, 5595		
<code>\@@_true_c:</code>	203, 219, 1748, 2096		
<code>\l_@@_type_of_col_tl</code>	831, 832, 2771, 2773, 5965, 5966, 5967, 5975, 5980		
<code>\c_@@_types_of_matrix_seq</code>	6638, 6639, 6644, 6648		
<code>\@@_update_for_first_and_last_row:</code> ..	932, 997, 1285, 2666, 2711		
<code>\@@_use_arraybox_with_notes:</code> ...	1584, 2317		

\{ 298, 1751, 1932,
1957, 2749, 6016, 6400, 6819, 6905, 7058, 7161
\} 298, 1754, 1932,
1942, 2749, 6016, 6409, 6819, 6905, 7058, 7161
\| 2751, 6015

_ .. 6634, 6662, 6667, 6674, 6682, 6683, 6694,
6695, 6758, 6783, 6784, 6797, 6803, 6807,
6810, 6813, 6825, 6831, 6843, 6887, 6888,
6889, 6897, 6903, 6910, 6923, 6930, 6931, 6939

A

\A 6214
\aboverulesep 2292
\addtocounter 429
\alph 375
\anchor 2966, 2967
\arraybackslash 1848, 2132
\arraycolor 1401, 6634
\arraycolsep
.. 620, 622, 624, 1091, 1185, 1306, 1307,
1612, 1616, 2250, 2586, 2590, 2600, 4959, 4966
\arrayrulecolor 109
\arrayrulewidth
141, 146, 158, 550, 923, 1108, 1110, 1116,
1147, 1674, 1680, 1770, 1820, 2006, 2013,
2178, 2185, 2309, 2348, 2475, 2477, 2483,
2493, 2495, 2501, 2524, 2526, 2532, 2550,
2552, 2558, 2584, 2588, 2600, 2603, 4375,
4376, 4378, 4394, 4396, 4578, 4579, 4581,
4592, 4598, 4711, 4722, 4728, 4764, 5038,
5773, 5828, 5853, 5855, 5867, 6146, 6343, 6347
\arraystretch
1184, 3399, 3417, 5334, 5423, 5439, 6319, 6322
\AtBeginDocument 23, 28, 85, 101, 178,
184, 228, 379, 493, 495, 497, 509, 700, 716,
2934, 3789, 3919, 3996, 4074, 4107, 4937, 6248
\AutoNiceMatrix 6017
\AutoNiceMatrixWithDelims ... 5969, 6009, 6021

B

\baselineskip 112, 118, 1875
\begingroup 2048
\bgroup 1474
\bigskip 1524, 3014
\Block 1240, 6856, 6863, 6910, 6951
\BNiceMatrix 5955
\bNiceMatrix 5952
\Body 1323

bool commands:

\bool_do_until:Nn 3067, 3135
\bool_gset_false:N 961,
1008, 1009, 1130, 1257, 1409, 1481, 1506,
1670, 2678, 2725, 4780, 4791, 4802, 4813, 5366
\bool_gset_true:N
1509, 1830, 2468, 2642, 2687, 2778, 3807,
3823, 3839, 3863, 3886, 3897, 4071, 4498, 4634
\bool_if:NnTF 169,
702, 707, 718, 723, 890, 893, 989, 1104,
1131, 1178, 1182, 1187, 1248, 1249, 1275,
1280, 1294, 1300, 1386, 1388, 1411, 1414,
1416, 1436, 1473, 1488, 1498, 1554, 1632,
1637, 1656, 1661, 1689, 1701, 1896, 1912,

1941, 2158, 2288, 2454, 2471, 2489, 2507,
2520, 2546, 2580, 2612, 2617, 2646, 2664,
2692, 2709, 2799, 2801, 2803, 2836, 2851,
2873, 2890, 2893, 2907, 3388, 3390, 3537,
3585, 3805, 3821, 3837, 3861, 3884, 4261,
4284, 4858, 4876, 4894, 5021, 5031, 5053,
5337, 5361, 5426, 5442, 5541, 5611, 5628,
5672, 5698, 5735, 6028, 6610, 6620, 6653, 6805
\bool_if:nTF
..... 186, 381, 408, 1067, 1564, 3221,
4096, 4338, 4608, 4612, 4738, 4742, 5047,
5296, 5725, 6125, 6135, 6137, 6150, 6156, 6165
\bool_lazy_all:nTF 1705,
1717, 2571, 2860, 4569, 4773, 4784, 4795, 4806
\bool_lazy_and:nnTF 231,
1774, 2247, 2510, 2585, 2589, 2597, 2647,
3377, 3644, 3903, 4349, 4704, 5791, 6369, 6386
\bool_lazy_or:nnTF
..... 530, 1001, 1059, 1697, 2230, 2259,
2337, 2695, 3454, 3702, 4330, 4362, 4366,
4416, 4420, 4850, 4868, 4886, 5271, 5276, 6272
\bool_lazy_or_p:nn 2650
\bool_not_p:n
... 1708, 1710, 1720, 1722, 2512, 2574, 2576
\bool_set:Nn 3458, 4278
\c_false_bool
1952, 1960, 1973, 1979, 1985, 3801, 3817, 3833
\g_tmpa_bool
..... 4498, 4507, 4541, 4549, 4554, 4634,
4642, 4676, 4684, 4689, 4780, 4791, 4802, 4813
\g_tmpb_bool 1670, 1701, 1830
\l_tmpb_bool ... 4855, 4869, 4887, 4909, 4922
\c_true_bool 1931

box commands:

\box_clear_new:N 1180, 1299
\box_dp:N
... 917, 937, 974, 994, 1049, 1208, 1217,
1290, 2137, 2199, 2367, 2380, 3417, 5396, 6322
\box_gclear_new:N 5324
\box_grotate:Nn 5363
\box_ht:N 918, 939, 945, 957, 980, 992, 1041,
1210, 1212, 1215, 1288, 1844, 1866, 1868,
1874, 2133, 2198, 2367, 2380, 3399, 5387, 6319
\box_move_down:nn 1049, 1872
\box_move_up:nn
..... 76, 78, 80, 1041, 2242, 2315, 2354
\box_rotate:Nn 951
\box_set_dp:Nn 973, 993, 2199
\box_set_ht:Nn 979, 991, 2198
\box_set_wd:Nn 967, 2249
\box_use:N 432, 958, 1048, 1877
\box_use_drop:N
..... 999, 1005, 1024, 1898, 2160,
2201, 2242, 2243, 2255, 2673, 5406, 5720, 5757
\box_wd:N 433, 968, 996,
1003, 1062, 1311, 1313, 1542, 2250, 2252,
2374, 2386, 2668, 2671, 2713, 2717, 5375, 6035
\l_tmpa_box
.. 415, 432, 433, 1310, 1311, 1312, 1313,
1599, 2198, 2199, 2201, 2242, 2243, 2367, 2380
\l_tmpb_box 2360, 2374, 2375, 2386

C

\c 59, 1693

2829, 2831, 2902, 4447, 4459, 4837, 4941, 4942, 5342, 5352, 5430, 5447, 5578, 5979, 6455	
\exp_not:n	1073, 1653, 2903, 3933, 3934, 4010, 4436, 4447, 4449, 4460, 5321, 5404, 5416, 5417, 5546, 5556, 5568, 5580, 5607, 5981, 6045, 6046
\ExplSyntaxOff	67, 1655, 5040
\ExplSyntaxOn	64, 1648, 5033
\extrarowheight	3399, 5335, 5424, 5440, 6319
F	
\fi	123, 1667, 2049, 2052, 4752, 4769
fi commands:	
\fi:	286
\five	2961, 2966
flag commands:	
\flag_clear_new:n	6448
\flag_height:n	6473
\flag_raise:n	6472
\fontdimen	2238
fp commands:	
\fp_eval:n	3657
\fp_to_dim:n	3692
\futurelet	128
G	
\globaldefs	1643, 5524
group commands:	
\group_insert_after:N	6029, 6030, 6032, 6033
H	
\halign	1219
\hbox	1102, 1608, 2315, 2354, 2473, 2491, 2518, 2522, 2548, 2582, 2987, 3000
hbox commands:	
\hbox:n	76, 78, 81, 2253
\hbox_gset:Nn	5326
\hbox_overlap_left:n	1042, 1050, 2452, 2669
\hbox_overlap_right:n	432, 2614, 2715
\hbox_set:Nn	415, 1039, 1310, 1312, 1599, 1870, 2036, 2360, 2375, 5610
\hbox_set:Nw	889, 1315, 1887, 2149, 2644, 2690
\hbox_set_end:	987, 1534, 1895, 2157, 2663, 2708
\hbox_to_wd:nn	457, 482
\Hdotsfor	1237, 6662, 6843
\hdotsfor	1168
\hdottedline	1234
\heavyrulewidth	2293
\hfil	1747, 2095
\hfill	141, 158
\Hline	1235, 4755
\hline	121
\hrule	125, 141, 158, 1147, 2293, 6160, 6507, 6531
\hskip	124
\Hspace	1236
\hspace	3898
\hss	1747, 2095
I	
\ialign	1099, 1195, 1218, 5503
\iddots	1233, 3869, 3870, 3875, 3876
\iddots	71, 1166, 1225
if commands:	
\if_mode_math:	284
\IfBooleanTF	1525
\ifnum	123, 4752, 4769
\ifstandalone	1494
\ignorespaces	1361, 2078
int commands:	
\int_case:nnTF	3843, 3849, 3867, 3873
\int_compare:nNnTF	136, 137, 153, 887, 888, 898, 905, 942, 952, 1144, 1146, 1272, 1278, 1283, 1535, 1552, 1556, 1567, 1571, 1572, 1639, 1865, 2058, 2076, 2271, 2310, 2349, 2421, 2449, 2693, 3077, 3084, 3088, 3090, 3145, 3152, 3156, 3158, 3321, 3360, 3441, 3476, 3478, 3986, 4061, 4325, 4370, 4388, 4424, 4455, 4472, 4473, 4480, 4481, 4485, 4915, 4917, 4919, 4921, 5330, 5332, 5368, 5380, 5691, 5723, 5727, 5800, 5802, 5983, 5985, 5987, 5991, 5993, 5995, 5997, 5999, 6353, 6355
\int_compare_p:n	3223, 3224, 3225, 3226, 4340, 4342, 5792, 5793
\int_do_until:nNnn	4281
\int_gadd:Nn	2075
\int_gincr:N	886, 915, 1497, 1794, 1858, 1901, 1907, 2026, 2041, 2544, 2569, 2688, 3539, 3587, 5018, 5311
\int_if_even:nTF	4229, 6473
\int_if_odd_p:n	4278
\int_incr:N	411, 1804
\int_min:nn	3024, 3026, 3028, 3030, 3040, 3042, 3231, 3232
\int_step_inline:nn	3022, 4225, 4227, 6366, 6383
\int_step_inline:nnnn	4847, 4865, 4880, 4883
\c_zero_int	1913, 4132, 4331
iow commands:	
\iow_now:Nn	62, 88, 1648, 1649, 1650, 1655
\iow_shipout:Nn	5033, 5034, 5040
\item	2276, 2282
K	
\kern	81
keys commands:	
\keys_define:nn	526, 546, 553, 631, 678, 730, 737, 781, 823, 837, 854, 867, 1329, 1419, 3889, 4236, 4245, 5007, 5237, 5459, 5817, 5923, 5963, 6087, 6092, 6172, 6193, 6202, 6571
\l_keys_key_str	6547, 6736, 6743, 6748, 6836, 6945, 6950, 6960, 6975, 6996, 7014, 7057, 7109, 7160
\keys_set:nn	200, 556, 558, 572, 812, 815, 822, 1359, 1423, 1431, 1519, 1520, 2772, 2782, 2791, 2913, 3324, 3363, 3444, 3513, 3565, 3989, 4064, 4081, 4240, 4259, 5020, 5540, 6094, 6098, 6225, 6278
\keys_set_known:nn	3855, 3879, 5288, 5774, 5829, 5868
\keys_set_known:nnN	5976
\l_keys_value_tl	6789, 6871, 6878, 7210
L	
\Ldots	1229, 3796, 3799

\ldots 1162, 1221
\leaders 141, 158
\left 1604, 2363, 2378, 6156, 6503, 6527
legacy commands:
\legacy_if:nTF 662
\line 2885, 6819, 6971

M

\makebox 1898, 2160
\mathinner 73
mode commands:
\mode_leave_vertical: 1486, 2131
msg commands:
\msg_error:nn 12
\msg_error:nnn 13
\msg_error:nnnn 14, 5521, 5528, 5532
\msg_fatal:nn 15
\msg_fatal:nnn 16
\msg_new:nnn 17
\msg_new:nnnn 18
\msg_redirect_name:nnn 20
\multicolumn 1239, 3900, 3909, 3915, 3937
\multispan 137, 138, 154, 155, 2047
\myfiledate 6
\myfileversion 7

N

\newcolumntype 255, 256, 257
\newcounter 371
\NewDocumentCommand 383,
406, 821, 1356, 1429, 2912, 3793, 3809,
3825, 3841, 3865, 3923, 4000, 4078, 4158,
4173, 4188, 4197, 4218, 4223, 4238, 4254,
4430, 4441, 4453, 5762, 5969, 6017, 6232, 6252
\NewDocumentEnvironment 1469,
2393, 2402, 2728, 2738, 2768, 2779, 2787, 5016
\NewExpandableDocumentCommand 261, 5258
\newlist 387, 398
\NiceArray 2784, 2793
\NiceArrayWithDelims 2733, 2743
nicematrix commands:
\g_nicematrix_code_after_tl .. 300, 671,
2417, 2891, 2894, 5543, 5553, 5565, 6105, 6112
\g_nicematrix_code_before_tl
1264, 2896, 2903, 4434, 4445, 4457, 5576, 5588
\NiceMatrixLastEnv 261
\NiceMatrixOptions 821, 7015, 7215
\NiceMatrixoptions 6875
\noalign 112, 118, 123, 146, 1126, 1201, 4752, 4929
\nobreak 401
\normalbaselines 1181
\NotEmpty 1246
\null 2074
\nulldelimiterspace 2374, 2386, 6141, 6345
\nullfont 6152, 6159, 6499, 6506, 6523, 6530
\numexpr 166, 167

O

\omit 136, 2451, 2467, 2543, 2568, 6102
\OnlyMainNiceMatrix 1242, 4464

P

\par 2270, 2278
\path 5935

peek commands:

\peek_meaning:NTF 171, 413, 1156
\peek_meaning_ignore_spaces:NTF 2395, 4755
\peek_meaning_remove_ignore_spaces:NTF 161
\peek_remove_spaces:n
..... 4432, 4443, 5260, 6234, 6254
\pgfdeclareshape 2959
\pgfextracty 5694
\pgfgetlastxy 475
\pgfpathcircle 3781
\pgfpathlineto 4589, 4595, 4719, 4725,
5849, 5860, 5907, 5920, 6070, 6360, 6377, 6411
\pgfpathmoveto 4588, 4594, 4718, 4724,
5848, 5859, 5906, 5919, 6065, 6359, 6376, 6402
\pgfpathrectanglecorners 4398, 4582, 4712, 5808
\pgfpointhead 473
\pgfpointheadanchor 176, 264, 2982,
2995, 3247, 3258, 3275, 3297, 3405, 3423,
5079, 5087, 5638, 5656, 5676, 5678, 5695,
5732, 6133, 6292, 6299, 6328, 6335, 6447, 6451
\pgfpointheaddiff .. 474, 1377, 1383, 1448, 1462, 1463
\pgfpointheadlineatime 3651
\pgfpointheadorigin 2458, 2625, 2967
\pgfpointheadscale 473
\pgfpointheadshapeborder 4119, 4122
\pgfrememberpicturepositiononpagetrue ...
920, 1015, 1114, 2457, 2481, 2499, 2530,
2556, 2594, 2623, 2945, 2972, 3021, 3608,
4118, 4561, 4696, 4950, 4995, 5122, 5132,
5143, 5613, 5776, 5837, 5884, 6060, 6119, 6280

\pgfscope 2980, 2993, 3648, 6077
\pgfset 442, 467, 1016, 5709, 5746, 6076, 6140, 6282
\pgfsetbaseline 1014
\pgfsetcornersarced 4397, 5784
\pgfsetlinewidth
..... 4598, 4728, 5811, 5840, 5887, 6347
\pgfsetrectcap 4599, 4729
\pgfsetroundcap 6073
\pgfsetstrokecolor 5782
\pgfsyspdfmark 65, 66
\pgftransformrotate 3655
\pgftransformshift 448, 473, 2981, 2994, 3032,
3649, 5708, 5730, 6078, 6082, 6142, 6425, 6435
\pgfusepath
... 3675, 3685, 4149, 4301, 4313, 4585, 5812
\pgfusepathqfill 3787, 4715
\pgfusepathqstroke 4600, 4730,
5850, 5861, 5908, 5921, 6074, 6361, 6378, 6412
\phantom 3806, 3822, 3838, 3862, 3885
\pNiceMatrix 5943

prg commands:

\prg_do_nothing:
..... 69, 187, 230, 239, 248, 543, 1201, 2888
\prg_new_conditional:Nnn 4328, 4336
\prg_replicate:nn 421, 2539,
2540, 3937, 4590, 4720, 5986, 5989, 5992, 5998
\prg_return_false: 4333, 4345
\prg_return_true: 4334, 4344
\ProcessKeysOptions 6591
\ProvideDocumentCommand 71
\ProvidesExplPackage 4

Q

\quad 402

quark commands:

`\q_stop` 133, 134,
150, 151, 172, 174, 1412, 1434, 1523, 1686,
1759, 1829, 1945, 1967, 2054, 2100, 2412,
2415, 4072, 4086, 4087, 4153, 4208, 4213,
4273, 4360, 4361, 4383, 4384, 4414, 4415,
5216, 5223, 5263, 5264, 5268, 5575, 5767,
5790, 5799, 5830, 5833, 5877, 5880, 5958,
5973, 6236, 6241, 6266, 6269, 6349, 6458, 6466

R

`\rectanglecolor` 1397, 2898, 4447
`\refstepcounter` 430
regex commands:

`\regex_const:Nn` 59
`\regex_match:nnTF` 6214
`\regex_replace_all:NnN` 1691
`\relax` 166, 167
`\renewcommand` 193, 213
`\RenewDocumentEnvironment`
..... 5942, 5945, 5948, 5951, 5954
`\RequirePackage` 1, 3, 9, 10, 11
`\right` 1622, 2370, 2382, 6165, 6511, 6535
`\rotate` 1241
`\roundedrectanglecolor` 1398, 5578
`\rowcolor` 1173, 1399
`\rowcolors` 1400
`\RowStyle` 1247

S

`\savedanchor` 2961
`\savenotes` 1473
scan commands:
`\scan_stop:` 2892
`\scriptstyle`
..... 893, 2646, 2692, 3636, 3637, 3671, 3681

seq commands:

`\seq_clear:N` 1684
`\seq_clear_new:N` 4817
`\seq_count:N` 2422, 4135
`\seq_gclear:N` 1250, 1251,
1293, 1295, 1500, 1501, 1502, 1503, 2298, 2889
`\seq_gclear_new:N` 1296, 1297, 1408, 2418, 2434
`\seq_gpop_left:NN` 2424, 2436
`\seq_gput_left:Nn` 667, 2060, 2062, 5538
`\seq_gput_right:Nn` 412, 1767, 2063,
3202, 4134, 5401, 5413, 5560, 6048, 6220, 6242
`\seq_gset_from_clist:Nn`
..... 2807, 2821, 2829, 2831
`\seq_gset_map_x:NNn` 2916
`\seq_gset_split:Nnn` 2420, 2435
`\seq_if_empty:NTF` 2257, 2817, 2825, 4833, 5586
`\seq_if_empty_p:N` ... 2862, 2863, 2864, 4350
`\seq_if_in:NnTF`
..... 665, 4390, 4538, 4544, 4551, 4673,
4679, 4686, 5082, 5090, 5636, 5654, 6216, 6648
`\seq_item:Nn` 1276, 1281, 1365, 1366, 1367, 1368
`\seq_map_function:NN` 2426
`\seq_map_indexed_inline:Nn` 4130, 4144
`\seq_map_inline:Nn`
..... 2276, 2282, 2438, 3216, 4288, 4499,
4501, 4503, 4635, 4637, 4639, 4910, 5504, 6351
`\seq_mapthread_function:NNN` 5211

`\seq_new:N` 274, 289, 324, 325, 326,
327, 328, 329, 330, 331, 336, 340, 372, 6638
`\seq_put_right:Nn` 4897, 5463
`\seq_set_eq:NN` 4263
`\seq_set_filter:NNn` 4264, 4286
`\seq_set_from_clist:Nn` 4837, 6639
`\seq_set_map_x:NNn` 6644
`\seq_use:Nn` 5595
`\seq_use:Nnnn`
..... 2822, 2830, 2832, 4838, 6832, 7220
`\l_tmpa_seq` 4264, 4286
`\l_tmpb_seq` 4263, 4264, 4286, 4288
`\setlist` 388, 399, 703, 708, 719, 724

siunitx commands:

`\siunitx_cell_begin:w` 189, 201, 233
`\siunitx_cell_end:` 204

skip commands:

`\skip_gadd:Nn` 2515
`\skip_gset:Nn` 2506
`\skip_gset_eq:NN` 2513, 2514
`\skip_horizontal:N` 142, 159, 1316,
1317, 1532, 1533, 1574, 1575, 1611, 1612,
1615, 1616, 1634, 1635, 1674, 1680, 1770,
1993, 2006, 2013, 2178, 2185, 2387, 2388,
2390, 2391, 2462, 2463, 2475, 2477, 2493,
2495, 2517, 2524, 2526, 2545, 2550, 2552,
2570, 2579, 2584, 2586, 2588, 2590, 2616,
2674, 2675, 2676, 2679, 2714, 2719, 2720, 2721
`\skip_horizontal:n` 433, 1062, 1816
`\skip_vertical:N`
146, 1108, 1110, 1607, 1618, 2267, 2292, 4929
`\skip_vertical:n` 957, 4762
`\g_tmpa_skip`
... 2506, 2513, 2514, 2515, 2517, 2545, 2570
`\c_zero_skip` 1206
`\smash` 6536, 6537
`\space` 297, 298
`\stepcounter` 419, 424

str commands:

`\c_backslash_str` 297
`\c_colon_str` 820
`\str_case:nn` 5701, 5713, 5738, 5750, 6396, 6405
`\str_case:nnTF`
... 1579, 1734, 2082, 2224, 4820, 6470, 6483
`\str_clear_new:N` 6277
`\str_gclear:N` 2908
`\str_gset:Nn`
... 1483, 2732, 2741, 2770, 2781, 2789, 6008
`\str_if_empty:NTF`
..... 924, 1027, 1117, 1482, 2459,
2484, 2502, 2533, 2559, 2605, 2626, 2731,
2740, 3036, 3048, 5203, 5230, 6417, 6431, 6441
`\str_if_eq:nnTF` 96, 296, 1097,
1762, 1765, 1809, 1829, 1854, 1915, 1947,
1969, 2000, 2172, 2407, 2409, 2442, 5780, 6110
`\str_if_eq_p:nn` 532, 1698,
1699, 1775, 4364, 4368, 4418, 4422, 5273, 5278
`\str_if_in:NnTF` 2212, 2324
`\str_new:N` 292, 513, 819
`\str_range:Nnn` 2216, 2328
`\str_set:Nn` 664, 808, 6219
`\str_set_eq:NN` 668, 820
`\l_tmpa_str` 664, 665, 667, 668

<code>\tl_if_empty:nTF</code>	645, 783, 825, 841, 857, 875, 2404, 2431, 3325, 3364, 3445, 3514, 3566, 3990, 4065, 4082, 4296, 4308, 6211, 6468, 6536, 6661
<code>\tl_if_empty_p:N</code>	1709, 1721, 3645, 3646
<code>\tl_if_empty_p:n</code>	2261, 5303
<code>\tl_if_eq:NNTF</code>	3610
<code>\tl_if_eq:NnTF</code>	1136, 1671, 2003, 2175, 2200, 2319, 4617, 4747, 4969, 4971, 6365, 6382
<code>\tl_if_eq:nnTF</code>	657, 799, 1945, 1967, 4131
<code>\tl_if_exist:NNTF</code>	1507
<code>\tl_if_in:NnTF</code>	4272, 4359, 4382, 4413
<code>\tl_if_in:nnTF</code>	1932, 1942, 1957
<code>\tl_if_single_token:nTF</code>	595, 807
<code>\tl_if_single_token_p:n</code>	58
<code>\tl_item:Nn</code>	244, 245, 247
<code>\tl_map_inline:nn</code>	2405
<code>\tl_new:N</code>	243, 246, 269, 281, 288, 300, 301, 302, 305, 312, 314, 338, 339, 343, 345, 348, 373, 498, 502, 521, 523, 524
<code>\tl_put_left:Nn</code>	1179, 1417
<code>\tl_put_right:Nn</code>	647, 1189, 1285, 1325, 1516
<code>\tl_range:nnn</code>	96
<code>\tl_set:Nn</code>	244, 4214, 4215, 4274, 4290, 4369, 4371, 4387, 4389, 4423, 4425, 4494, 5289, 5801, 5803, 5834, 5835, 5881, 5882
<code>\tl_set_eq:NN</code>	500, 4211, 4212, 4372, 4510, 4645, 4973, 5003, 5285, 5831, 5832, 5878, 5879, 6239, 6240, 6267, 6268, 6270, 6271
<code>\tl_set_rescan:Nnn</code>	2419, 3792, 3922, 3999, 4077, 6251
<code>\tl_to_str:n</code>	6645
<code>\g_tmpa_tl</code>	241, 244, 247
tmpc commands:	
<code>\l_tmpc_dim</code>	319, 3029, 3033, 4375, 4376, 4399, 4568, 4579, 4584, 4589, 4595, 4703, 4714, 4719, 4725, 5620, 5626, 5670, 5798, 5809, 5904, 5907, 6067, 6070, 6078
<code>\l_tmpc_int</code>	4279, 4280, 4281
<code>\l_tmpc_tl</code>	4209, 4211, 4214, 4372, 4391, 4495, 4509, 4510, 4513, 4518, 4520, 4524, 4529, 4531, 4631, 4644, 4645, 4648, 4653, 4655, 4659, 4664, 4666, 5831, 5843, 5856, 5878, 5895, 5901, 6237, 6239, 6243
tmpd commands:	
<code>\l_tmpd_dim</code>	320, 3031, 3034, 4396, 4399, 4580, 4584, 4710, 4714, 5622, 5626, 5648, 5659, 5663, 5666, 5670, 5807, 5810, 6069, 6070, 6082
<code>\l_tmpd_tl</code>	4210, 4212, 4215, 5832, 5845, 5854, 5879, 5891, 5912, 6238, 6240, 6243
token commands:	
<code>\token_to_str:N</code>	6633, 6634, 6662, 6753, 6758, 6764, 6789, 6797, 6803, 6807, 6819, 6825, 6843, 6856, 6862, 6903, 6910, 6923, 6940, 6950, 6962, 6971, 6977, 7015, 7215
U	
<code>\unskip</code>	2286
use commands:	
<code>\use:N</code>	1070, 1320, 1321, 1510, 1529, 1530, 2755, 2775, 4147
<code>\use:n</code>	4083, 4464, 5340, 5350, 5428, 5445, 5977, 6454
<code>\usepackage</code>	6617, 6627
<code>\usepgfmodule</code>	2
V	
vbox commands:	
<code>\vbox:n</code>	81
<code>\vbox_set_top:Nn</code>	954
<code>\vbox_to_ht:nn</code>	453, 480, 2366, 2379
<code>\vbox_to_zero:n</code>	956
<code>\vcenter</code>	1605, 2364, 6157, 6504, 6528, 6940
<code>\Vdots</code>	1231, 3828, 3831
<code>\vdots</code>	1164, 1223
<code>\Vdotsfor</code>	1238
<code>\vfill</code>	456, 482
<code>\vline</code>	127
<code>\VNiceMatrix</code>	5949
<code>\vNiceMatrix</code>	5946
<code>\vrule</code>	125, 1844, 2133, 2137
<code>\vskip</code>	124
<code>\vtop</code>	1106
X	
<code>\xglobal</code>	901, 908, 2656, 2700
Z	
<code>\Z</code>	6214

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	7
5.1.1	The vertical rules	7
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	8
5.3	The tools of <code>nicematrix</code> for the rules	9
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The keys <code>hvlines</code> and <code>hvlines-except-borders</code>	9
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
6	The color of the rows and columns	11
6.1	Use of <code>colortbl</code>	11
6.2	The tools of <code>nicematrix</code> in the <code>\CodeBefore</code>	12
6.3	Color tools with the syntax of <code>colortbl</code>	15
7	The command <code>\RowStyle</code>	16
8	The width of the columns	16
9	The exterior rows and columns	17
10	The continuous dotted lines	19
10.1	The option <code>nullify-dots</code>	20
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	21
10.3	How to generate the continuous dotted lines transparently	22
10.4	The labels of the dotted lines	22
10.5	Customisation of the dotted lines	23
10.6	The dotted lines and the rules	24
11	The <code>\CodeAfter</code>	24
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	24
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	25
12	The notes in the tabulars	27
12.1	The footnotes	27
12.2	The notes of tabular	27
12.3	Customisation of the tabular notes	29
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	31
13	Other features	31
13.1	Use of the column type <code>S</code> of <code>siunitx</code>	31
13.2	Alignment option in <code>{NiceMatrix}</code>	31
13.3	The command <code>\rotate</code>	31
13.4	The option <code>small</code>	32
13.5	The counters <code>iRow</code> and <code>jCol</code>	33
13.6	The option <code>light-syntax</code>	33
13.7	Color of the delimiters	34
13.8	The environment <code>{NiceArrayWithDelims}</code>	34
14	Use of <code>Tikz</code> with <code>nicematrix</code>	34
14.1	The nodes corresponding to the contents of the cells	34
14.2	The “medium nodes” and the “large nodes”	35
14.3	The nodes which indicate the position of the rules	36
14.4	The nodes corresponding to the command <code>\SubMatrix</code>	37

15	API for the developpers	38
16	Technical remarks	38
16.1	Definition of new column types	38
16.2	Diagonal lines	39
16.3	The “empty” cells	39
16.4	The option <code>exterior-arraycolsep</code>	40
16.5	Incompatibilities	40
17	Examples	41
17.1	Utilisation of the key “tikz” of the command <code>\Block</code>	41
17.2	Notes in the tabulars	41
17.3	Dotted lines	42
17.4	Dotted lines which are no longer dotted	43
17.5	Stacks of matrices	44
17.6	How to highlight cells of a matrix	47
17.7	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	49
18	Implementation	49
19	History	212
	Index	219