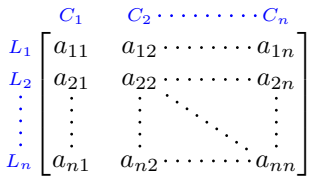
The package nicematrix*

F. Pantigny
fpantigny@wanadoo.fr

May 27, 2020

Abstract

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



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 as MiKTeX or TeXlive.

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

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

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

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

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

*This document corresponds to the version 4.1 of nicematrix, at the date of 2020/05/27.

1 The environments of this package

The package nicematrix defines the following new environments.

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

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

However, there are some small differences:

- For technical reasons, in the preamble of these environments, the user must use the letters L, C and R[1] instead of l, c and r, included in the commands \multicolumn and in the types of columns defined by \newcolumntype.

  * In {NiceArray} (and its variants), the columns of type w (ex. : wc{1cm}) are composed in math mode whereas, in {array} of array, they are composed in text mode.

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.

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

2 The vertical space between the rows

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

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

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

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. 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.

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

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

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

[1] The column types L, C and R are defined locally inside {NiceTabular} or {NiceArray} with \newcolumntype of array. This definition overrides an eventual previous definition.

3 The vertical position of the arrays

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

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

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

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`[2]).

```
\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
2.

n	0	1	2	3	4	5
u_n	1	2	4	8	16	32

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

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

1. an item
2.

n	0	1	2	3	4	5
u_n	1	2	4	8	16	32

4 The blocks

In the environments of nicematrix, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array. The command `\Block` don't create space by itself.

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

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

[2] It's also possible to use `\firsthline` in the environments of nicematrix.

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

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

It's also possible to use the command `\Block` in mathematical matrices.

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

$$\left[\begin{array}{ccc|c} \Block{3-3}{A} & & & 0 \\ & & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right]$$

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.

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

$$\left[\begin{array}{ccc|c} \Block{3-3}<\Large>{A} & & & 0 \\ & & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right]$$

5 The rules

The usual techniques for the rules may be used in the environments of nicematrix (excepted `\vline`).

5.1 The thickness and the color of the rules

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

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

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

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

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 22.

5.2 A remark about \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.

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	B	C	D

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

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

A	B	C	D
A	B	C	D

5.3 The keys hlines and vlines

The key `hlines` draws all the horizontal rules and the key `vlines` draws all the vertical rules. In fact, in the environments with delimiters (as `{pNiceMatrix}` or `{bNiceArray}`) the exteriors rules are not drawn (as expected).

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

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

However, there is a difference between the key `vlines` and the use of the specifier "`|`" in the preamble of the environment: the rules drawn by `vlines` completely cross the double-rules drawn by `\hline\hline` (you don't need hhline).

```
$\begin{NiceMatrix}[vlines] \hline
a & b & c & d \\ \hline \hline
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\ \hline
\end{NiceMatrix}$
```

a	b	c	d
1	2	3	4
1	2	3	4

If you use booktabs (which provides `\toprule`, `midrule`, `bottomrule`, etc.) and you really want to draw vertical rules (something opposed to the spirit of booktabs), you should remark that the key `vlines` in compatible with booktabs.

```
$\begin{NiceMatrix}[vlines] \toprule
a & b & c & d \\ \midrule
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\ \bottomrule
\end{NiceMatrix}$
```

a	b	c	d
1	2	3	4
1	2	3	4

5.4 The key hvlines

The key `hvlines` draws all the vertical and horizontal rules *excepted in the blocks.*[3]

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

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

5.5 La commande \diagbox

The command `\diagbox` (inspired by the package diagbox), allows, when it is used in a cell, to slash that cell diagonally downwards.[4].

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

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

5.6 Dotted rules

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

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

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

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`.

Remark : In the package array (on which the package nicematrix relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule[5]. In nicematrix, the dotted lines drawn by `\hdottedline` and ":" do likewise.

[3] In fact, when the key `hvlines` is in force, the rules are also not drawn in the virtual blocks delimited by cells relied par dotted lines (cf. p. 15).

[4] The author of this document considers that type of construction as graphically poor.

[5] In fact, this is true only for `\hline` and "|" but not for `\cline`.

6 The color of the rows and columns

With the classical package colortbl, it's possible to color the cells, rows and columns of a tabular. However, the resulting PDF is not always perfectly displayed by the PDF viewers, in particular in conjonction with rules. With some PDF viewers, some vertical rules seem to vanish. On the other side, some thin horizontal white lines may appear in some circonstances.

The package nicematrix provides similar tools which do not present these drawbacks. It provides a key `code-before`[6] for some code which will be executed *before* the drawing of the tabular. In this `code-before`, new commands are available: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors`.

These commands are independent of colortbl.[7]

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

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

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

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

a	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|}[code-before =
\rectanglecolor{blue!15}{2-2}{3-3}]
\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 `\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).

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

a_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}

[6] There is also a key `code-after` : see p. 16.

[7] Thus, it's possible to coloror the rules, the cells, the rows, the columns, etc. without loading colortbl.

- The command `\columncolor` takes its name from the command `\columncolor` of colortbl. Its syntax is similar to the syntaxe of `\rowcolor`.

- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of xcolor[8]. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular, beginning with the row whose number is given in first (mandatory) argument. The two other (mandatory) arguments are the colors.

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

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

- The command `\chessboardcolors` takes in mandatory arguments two colors and colors the cells of the tabular in quincunx with these colors.

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

$$\begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$$

We have used the key `R` which aligns all the columns rightwards (cf. p. 16).

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

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

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.

7 The width of the columns

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

[8] The command `\rowcolors` of color is available when xcolor is loaded with the option `table`.

{pNiceArray}, etc., they are composed in math mode (whereas, in {array} of array, they are composed in text mode).

```
\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.[9]

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

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

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.

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

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

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[10]. The environment {NiceMatrixBlock} has no direct link with the command \Block presented previously in this document (cf. p. 3).

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

$$\begin{bmatrix} 9 & 17 \\ -2 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1245345 \\ 345 & 2 \end{bmatrix}$$

Several compilations may be necessary to achieve the job.

[9] The result is achieved with only one compilation (but Tikz will have written informations in the .aux file and a message requiring a second compilation will appear).

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

8 The exterior rows and columns

The options `first-row`, `last-row`, `first-col` and `last-col` allow the composition of exterior rows and columns in the environments of nicematrix.
A potential "first row" (exterior) has the number 0 (and not 1). Idem for the potential "first column".

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

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

The dotted lines have been drawn with the tools presented p. 11.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `R` for the first column and `L` for the last one.
- One may wonder how nicematrix determines the number of rows and columns which are needed for the composition of the "last row" and "last column".
  - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
  - When the option `light-syntax` (cf. p. 18) 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}}
```

```
$\begin{pNiceArray}{CC|CC}[first-row,last-row=5,first-col,last-col,nullify-dots]
       & C_1    & \Cdots &        & C_4    &        \\
L_1    & a_{11} & a_{12} & a_{13} & a_{14} & L_1    \\
\Vdots & a_{21} & a_{22} & a_{23} & a_{24} & \Vdots \\
\hline
       & a_{31} & a_{32} & a_{33} & a_{34} &        \\
L_4    & a_{41} & a_{42} & a_{43} & a_{44} & L_4    \\
       & C_1    & \Cdots &        & C_4    &
\end{pNiceArray}$
```

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

Remarks

- As shown in the previous example, the horizontal and 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 22.

- 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. 8) doesn't apply to the "first column" and "last column".

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

9 The continuous dotted lines

Inside the environments of the package nicematrix, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`.[11]

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells[12] 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`.[13]

```
\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}
```

$$\begin{bmatrix} a_1 & \cdots & & a_1 \\ \vdots & & & \\ & a_2 & \cdots & a_2 \\ \vdots & \vdots & \iddots & \\ a_1 & a_2 & & a_n \end{bmatrix}$$

[11] The command `\iddots`, defined in nicematrix, is a variant of `\ddots` with dots going forward. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of unicode-math.

[12] The precise definition of a "non-empty cell" is given below (cf. p. 23).

[13] It's also possible to change the color of all theses dotted lines with the option `xdots/color` (*xdots* to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 14.

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}
```

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

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 only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF[14]).

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

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.[15]
However, a command `\hspace*` might interfer with the construction of the dotted lines. That's why the package nicematrix provides a command `\Hspace` which is a variant of `\hspace` transparent for the dotted lines of nicematrix.

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

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

9.1 The option nullify-dots

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

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

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

[14] And it's not possible to draw a `\Ldots` and a `\Cdots` line between the same cells.

[15] In nicematrix, one should use `\hspace*` and not `\hspace` for such an usage because nicematrix loads array. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 8

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

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

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

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

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

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

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

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

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

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

9.2 The command \Hdotsfor

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.

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

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

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

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

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

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

9.3 How to generate the continuous dotted lines transparently

The package nicematrix provides an option called `transparent` for using existing code transparently in the environments of the amsmath : `{matrix}`, `{pmatrix}`, `{bmatrix}`, etc. In fact, this option is an alias for the conjonction of two options: `renew-dots` and `renew-matrix`.[16]

[16] The options `renew-dots`, `renew-matrix` and `transparent` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage` (it's an exception for these three specific options.)

- The option `renew-dots`

  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`[11] 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 option `transparent`, a classical code gives directly the ouput of nicematrix.

```
\NiceMatrixOptions{transparent}
\begin{pmatrix}
1      & \cdots & \cdots & 1      \\
0      & \ddots &        & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0      & \cdots & 0      & 1
\end{pmatrix}
```

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

9.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `code-after` which is described p. 16) 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`.

```
$\begin{bNiceMatrix}
1 & \hspace*{1cm}             & 0 \\[8mm]
  & \Ddots^{n \text{ times}} &   \\
0 &                           & 1
\end{bNiceMatrix}$
```

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

9.5 Customization of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and by the command `\line` in the `code-after` which is described p. 16) may be customized by three options (specified between square brackets after the command):

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

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

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

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

The option xdots/color

The option `xdots/color` fixes the color or the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 10.

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 recommanded 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).[17]

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

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

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

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

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

9.6 The dotted lines and the key hvlines

We have said (cf. p. 6) that the key `hvlines` draws all the horizontal and vertical rules, excepted in the blocks. In fact, when this key is in force, the rules are also not drawn in the virtual blocks delimited by cells relied par dotted lines.

```
\NiceMatrixOptions{nullify-dots}
$\begin{pNiceMatrix}[rules/color=gray,hvlines,margin]
0      & \Cdots &  &  &        & 0      \\
1      & \Cdots &  &  & 1      & 2      \\
0      & \Ddots &  &  & \Vdots & \Vdots \\
\Vdots & \Ddots &  &  &        &        \\
       &        &  &  &        &        \\
0      & \Cdots &  & 0 &  1    & 2      \\
\end{pNiceMatrix}$
```

$$\left(\begin{array}{cccc|c} 0 & \cdots & \cdots & 0 & 0 \\ \hline 1 & \cdots & \cdots & 1 & 2 \\ 0 & \ddots & & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & 2 \end{array}\right)$$

[17] The first reason of this behaviour is that the PDF format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the PDF readers. It's easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded dots needs a specification of each dot in the PDF file.

10 The code-after

The option `code-after` may be used to give some code that will be excuted after the construction of the matrix.[18]
A special command, called `\line`, is available to draw directly dotted lines between nodes. It takes two arguments for the two cells to rely, both of the form i-j where is the number of row and j is the number of column. It may be used, for example, to draw a dotted line between two adjacents cells.

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

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

For the readability 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`. For an example, cf. p. 27.

11 Other features

11.1 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.

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

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

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

11.2 Alignement option in {NiceMatrix}

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

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

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

There is also a key `S` which sets all the columns all type `S` of siunitx (if this package is loaded).[19]

[18] There is also a key `code-before` described p. 7.
[19] This is a part of the functionality provided by the environments `{pmatrix*}`, `{bmatrix*}`, etc. of mathtools.

11.3 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`.

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

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

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

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

11.4 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).

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

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

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.

11.5 The counters iRow and jCol

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column[20]. Of course, the user must not change the value of these counters which are used internally by nicematrix.
In the `code-before` (cf. p. 7) and in the `code-after` (cf. p. 16), `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).

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

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

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

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

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

11.6 The option light-syntax

The option `light-syntax` (inpired by the package spalign) allows the user to compose the arrays with a lighter syntax, which gives a better legibility of the TeX source.
When this option is used, one should use the semicolon for the end of a row and spaces or tabulations to separate the columns. However, as usual in the TeX world, the spaces after a control sequence are discarded and the elements between curly braces are considered as a whole.

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

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

$$\begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} 2\cos a & \cos a + \cos b \\ \cos a + \cos b & 2\cos b \end{bmatrix} \end{array}$$

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.[21]

[20] We recall that the exterior "first row" (if it exists) has the number 0 and that the exterior "first column" (if it exists) has also the number 0.

[21] The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX

11.7 The environment {NiceArrayWithDelims}

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

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

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

12 Utilisation of Tikz with nicematrix

12.1 The nodes corresponding to the contents of the cells

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

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

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

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

Don't forget the options `remember picture` and `overlay`.

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

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

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

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 27).

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.

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

12.2 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`.[22]

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.[23]

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

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`.[24]

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

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

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

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

[22] There is also an option `create-extra-nodes` which is an alias for the conjonction of `create-medium-nodes` and `create-large-nodes`.

[23] There is no "large nodes" created in the exterior rows and columns (for these rows and columns, cf. p. 10).

[24] The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

Here is a array composed with the following code:

```
\large
\begin{NiceTabular}{wl{2cm}LL}[hvlines]
fraise & amande & abricot \\
prune & pêche & poire  \\[1ex]
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`.

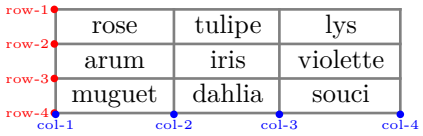
fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

Here are the "large nodes" of this array (without utilisation of `margin` nor `extra-margin`).

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

12.3 The "row-nodes" and the "col-nodes"

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



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

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

$$
\begin{array}{ccccccccc}
1 & & & & & & & & \\
1 & 1 & & & & & & & \\
1 & 2 & 1 & & & & & & \\
1 & 3 & 3 & 1 & & & & & \\
1 & 4 & 6 & 4 & 1 & & & & \\
1 & 5 & 10 & 10 & 5 & 1 & & & \\
1 & 6 & 15 & 20 & 15 & 6 & 1 & & \\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 & \\
1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1
\end{array}
$$

13 Technical remarks

13.1 Definition of new column types

The package nicematrix provides the command `\OnlyMainNiceMatrix` which is meant to be used in definitions of new column types. Its argument is evaluated if and only if we are in the main part of the array, that is to say not in an eventual exterior row.
For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job[25]:

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

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

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

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

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

13.2 Diagonal lines

By default, all the diagonal lines[26] of a same array are "parallelized". That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\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):

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

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

[25] The command `\vrule` is a TeX (and not LaTeX) command.
[26] We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

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

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

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

The same example without parallelization:

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

13.3 The "empty" cells

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

For nicematrix, the precise rules are as follow.

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

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

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

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

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

13.4 The option exterior-arraycolsep

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

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

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

13.5 Incompatibilities

The package nicematrix is not compatible with threeparttable.

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

14 Examples

14.1 Dotted lines

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

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

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

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

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

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

An example with `\multicolumn`:

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

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

An example with `\Hdotsfor`:

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

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

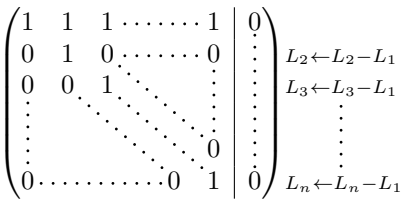
An example for the resultant of two polynoms:

```
\setlength{\extrarowheight}{1mm}
\[\begin{vNiceArray}{CCCC:CCC}[columns-width=6mm]
a_0   &      &&       &b_0    &      &     \\
a_1   &\Ddots&&       &b_1    &\Ddots&     \\
\Vdots&\Ddots&&       &\Vdots &\Ddots&b_0  \\
a_p   &      &&a_0    &       &      &b_1   \\
      &\Ddots&&a_1    &b_q    &      &\Vdots\\
      &      &&\Vdots &       &\Ddots&      \\
      &      &&a_p    &       &      &b_q
\end{vNiceArray}\]
```

$$\begin{vmatrix} a_0 & & & \vdots & b_0 & & \\ a_1 & \ddots & & & b_1 & \ddots & b_0 \\ \vdots & & \ddots & & \vdots & & b_1 \\ a_p & & a_0 & & b_q & & \\ & \ddots & a_1 & & & \ddots & \vdots \\ & & \vdots & & & & b_q \end{vmatrix}$$

An example for a linear system:

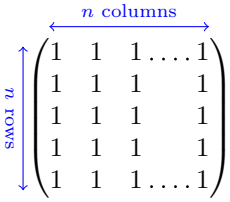
```
$\begin{pNiceArray}{*6C|C}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
1      & 1 & 1 &\Cdots &   & 1      & 0      & \\
0      & 1 & 0 &\Cdots &   & 0      &        & L_2 \gets L_2-L_1 \\
0      & 0 & 1 &\Ddots &   & \Vdots &        & L_3 \gets L_3-L_1 \\
       &   &   &\Ddots &   & 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



14.2 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
  {nullify-dots,code-for-first-col = \color{blue},code-for-first-col=\color{blue}}
$\begin{pNiceMatrix}[first-row,first-col]
       &   & \Ldots[line-style={solid,<->},shorten=0pt]^{n \text{ columns}} \\
       & 1 & 1 & 1 & \Ldots & 1 \\
       & 1 & 1 & 1 &  & 1 \\
\Vdots[line-style={solid,<->}]_{n \text{ rows}} & 1 & 1 & 1 &  & 1 \\
       & 1 & 1 & 1 &  & 1 \\
       & 1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$
```



14.3 Width of the columns

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

```
\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
 { last-col,code-for-last-col = \color{blue}\scriptstyle,light-syntax}
\setlength{\extrarowheight}{1mm}
$\begin{pNiceArray}{CCCC:C}
  1  1  1   1   1 ;
  2  4  8  16   9 ;
  3  9 27  81  36 ;
  4 16 64 256 100
\end{pNiceArray}$
\medskip
$\begin{pNiceArray}{CCCC:C}
  1  1  1   1  1 ;
  0  2  6  14  7        { L_2 \gets -2 L_1 + L_2 } ;
  0  6 24  78 33        { L_3 \gets -3 L_1 + L_3 } ;
  0 12 60 252 96        { L_4 \gets -4 L_1 + L_4 }
\end{pNiceArray}$
...
\end{NiceMatrixBlock}
```

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 9 \\ 3 & 9 & 27 & 81 & 36 \\ 4 & 16 & 64 & 256 & 100 \end{array}\right)$$

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 0 & 2 & 6 & 14 & 7 \\ 0 & 6 & 24 & 78 & 33 \\ 0 & 12 & 60 & 252 & 96 \end{array}\right) \begin{array}{l} \\ L_2 \leftarrow -2L_1+L_2 \\ L_3 \leftarrow -3L_1+L_3 \\ L_4 \leftarrow -4L_1+L_4 \end{array}$$

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 3 & 12 & 39 & \frac{33}{2} \\ 0 & 1 & 5 & 21 & 8 \end{array}\right) \begin{array}{l} \\ L_2 \leftarrow \frac{1}{2}L_2 \\ L_3 \leftarrow \frac{1}{2}L_3 \\ L_4 \leftarrow \frac{1}{12}L_4 \end{array}$$

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 3 & 18 & 6 \\ 0 & 0 & -2 & -14 & -\frac{9}{2} \end{array}\right) \begin{array}{l} \\ \\ L_3 \leftarrow -3L_2+L_3 \\ L_4 \leftarrow L_2-L_4 \end{array}$$

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & -2 & -14 & -\frac{9}{2} \end{array}\right) \begin{array}{l} \\ \\ L_3 \leftarrow \frac{1}{3}L_3 \\ \\ \end{array}$$

$$\left(\begin{array}{cccc:c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & 0 & -2 & -\frac{1}{2} \end{array}\right) \begin{array}{l} \\ \\ \\ L_4 \leftarrow L_3+L_4 \end{array}$$

14.4 How to highlight cells of the matrix

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

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

In order to highlight a cell of a matrix, it's possible to "draw" one of the correspondant nodes (the "normal node", the "medium node" or the "large node"). In the following example, we use the "large nodes" of the diagonal of the matrix (with the Tikz key "`name suffix`", it's easy to use the "large nodes").
We redraw the nodes with other nodes by using the Tikz library `fit`. Since we want to redraw the nodes exactly, we have to set `inner sep = 0 pt` (if we don't do that, the new nodes will be larger that the nodes created by nicematrix).

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

$$\left(\begin{array}{|c|ccc} \hline a_{11} & a_{12} & a_{13} & a_{14} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ \hline \end{array}\right)$$

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the command `\hline`, the specifier "`|`" and the options `hlines`, `vlines` and `hvlines` spread the cells.[29]

[29] For the command `\cline`, see the remark p. 5.

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

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

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

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

This code fails with `latex-dvips-ps2pdf` because Tikz for `dvips`, as for now, doesn't support blend modes. However, the following code, in the preamble, should activate blend modes in this way of compilation.

```
\ExplSyntaxOn
\makeatletter
\tl_set:Nn \l_tmpa_tl {pgfsys-dvips.def}
\tl_if_eq:NNT \l_tmpa_tl \pgfsysdriver
  {\cs_set:Npn\pgfsys@blend@mode#1{\special{ps:~/\tl_upper_case:n #1~.setblendmode}}}
\makeatother
\ExplSyntaxOff
```

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

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

$$\begin{pmatrix} \multicolumn{3}{c}{\multirow{3}{*}{A}} & 0 \\ & & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 0 \end{pmatrix}$$

Consider now the following matrix which we have named `example`.

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

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

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

```
\tikzset{mes-options/.style={remember picture,
                             overlay,
                             name prefix = exemple-,
                             highlight/.style = {fill = red!15,
                                                 blend mode = multiply,
                                                 inner sep = 0pt,
                                                 fit = #1}}}

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

We obtain the following matrix.

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

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

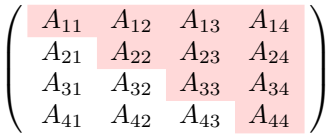
```
\begin{tikzpicture}[mes-options, name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
```

We obtain the following matrix.

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

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

```
\begin{pNiceArray}{>{\strut}CCCC}[create-large-nodes,margin,extra-margin=2pt]
  A_{11} & A_{12} & A_{13} & A_{14} \\
  A_{21} & A_{22} & A_{23} & A_{24} \\
  A_{31} & A_{32} & A_{33} & A_{34} \\
  A_{41} & A_{42} & A_{43} & A_{44}
\CodeAfter
  \tikz \path [name suffix = -large,fill = red!15, blend mode = multiply]
      (1-1.north west)
   |- (2-2.north west)
   |- (3-3.north west)
   |- (4-4.north west)
   |- (4-4.south east)
   |- (1-1.north west) ;
\end{pNiceArray}
```



14.5 Direct use of the Tikz nodes

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

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

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


\NiceMatrixOptions{nullify-dots}
```

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

```
$\begin{array}{cc}
&
```

The matrix B has a "first row" (for C_j) and that's why we use the key `first-row`.

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

The matrix A has a "first column" (for L_i) and that's why we use the key `first-col`.

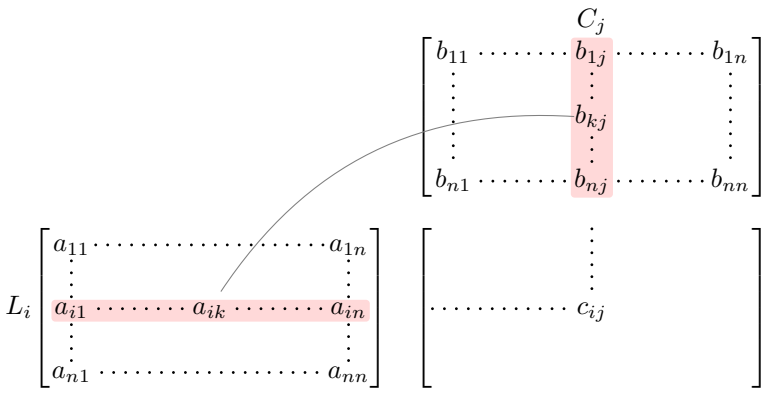
```
\begin{bNiceArray}{CC>{\strut}CCC}[name=A,first-col]
    & a_{11} & \Cdots &        &        & a_{1n} \\
    & \Vdots &        &        &        & \Vdots \\
L_i & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} \\
    & \Vdots &        &        &        & \Vdots \\
    & a_{n1} & \Cdots &        &   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

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

```
\begin{bNiceArray}{CC>{\strut}CCC}
       & &        & & \\
       & & \Vdots     \\
\Cdots & & c_{ij}     \\
\\
\\
\end{bNiceArray}
\end{array}$

\end{NiceMatrixBlock}

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



15 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: 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 registred 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`.

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

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

```
\RequirePackage{l3keys2e}
\ProvidesExplPackage
  {nicematrix}
  {\myfiledate}
  {\myfileversion}
  {Mathematical matrices with PGF/TikZ}
```

The version of 2020/02/08 of expl3 has replaced `\l_keys_key_tl` by `\l_keys_key_str`. We have immediately changed in this file. Now, you test the existence of `\l_keys_key_str` in order to detect whether the version of LaTeX used by the final user is up to date.

```
\msg_new:nnn { nicematrix } { expl3~too~old }
  {
    Your~version~of~LaTeX~(especially~expl3)~is~too~old.~
    You~can~go~on~but~you~will~probably~have~other~errors~
    if~you~use~the~functionalities~of~nicematrix.
  }
\cs_if_exist:NF \l_keys_key_str
  { \msg_error:nn { nicematrix } { expl3~too~old } }
```

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

We load some packages.

```
\RequirePackage { array }
\RequirePackage { amsmath }
\RequirePackage { xparse }
```

```
\cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
\cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
\cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_redirect_name:nn
  { \msg_redirect_name:nnn { nicematrix } }
```

Technical definitions

```
\bool_new:N \c_@@_booktabs_loaded_bool
\bool_new:N \c_@@_tikz_loaded_bool
\AtBeginDocument
  {
    \@ifpackageloaded { booktabs }
      { \bool_set_true:N \c_@@_booktabs_loaded_bool }
      { }
    \@ifpackageloaded { tikz }
      {
```

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`-`\endtikpicture` (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.

```
        \bool_set_true:N \c_@@_tikz_loaded_bool
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
      }
      {
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
      }
  }
```

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

```
\bool_new:N \c_@@_revtex_bool
\@ifclassloaded { revtex4-1 }
  { \bool_set_true:N \c_@@_revtex_bool }
  { }
\@ifclassloaded { revtex4-2 }
  { \bool_set_true:N \c_@@_revtex_bool }
  { }
```

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

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

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

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

```
\AtBeginDocument
  {
    \@ifpackageloaded { booktabs }
      { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
      { }
  }
\cs_set_protected:Npn \nicematrix@redefine@check@rerun
  {
    \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun
```

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

```
    \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
      {
        \str_set:Nx \l_tmpa_str { \tl_range:nnn { ##1 } 1 3 }
        \str_if_eq:VnF \l_tmpa_str { nm- }
          { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
      }
  }
```

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

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

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

```
        \cs_set_protected:Npn \CT@arc@ { }
        \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
        \cs_set:Npn \CT@arc #1 #2
          {
            \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
             { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
          }
        \cs_set:Npn \hline
          {
            \noalign { \ifnum 0 = `} \fi
            \cs_set_eq:NN \hskip \vskip
            \cs_set_eq:NN \vrule \hrule
            \cs_set_eq:NN \@width \@height
```

```
            { \CT@arc@ \vline }
            \futurelet \reserved@a
            \@xhline
          }
      }
  }
```

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

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

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

```
    \everycr { }
    \cr
    \noalign { \skip_vertical:N -\arrayrulewidth }
  }
```

The following version of `\cline` spreads the array of a quantity equal to `\arrayrulewidth` as does `\hline`. It will be loaded except if the key `standard-cline` has been used.

```
\cs_set:Npn \@@_cline
```

We have to act in a fully expandable way since there may be `\noalign` (in the `\multispan`) to detect. That's why we use `\exp_arg:Ne` and not `\exp_arg:Nx`.

```
  { \exp_args:Ne \@@_cline_i:nn \l_@@_first_col_int }
```

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

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

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

```
    \int_compare:nNnT { #1 } < { #2 }
      { \multispan { \int_eval:n { #2 - #1 } } & }
    \multispan { \int_eval:n { #3 - #2 + 1 } }
      { \CT@arc@ \leaders \hrule \@height \arrayrulewidth \hfill }
```

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

```
    \peek_meaning_remove_ignore_spaces:NTF \cline
      { & \exp_args:Ne \@@_cline_i:nn { \@@_succ:n { #3 } } }
      { \everycr { } \cr }
  }


```

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

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

The following command is a small shortcut.

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


\cs_new_protected:Npn \@@_set_CT@arc@:
```

```
  { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
\cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
  { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
\cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
  { \cs_set:Npn \CT@arc@ { \color { #1 } } }
```

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.

```
\bool_new:N \c_@@_siunitx_loaded_bool
\AtBeginDocument
  {
    \@ifpackageloaded { siunitx }
      { \bool_set_true:N \c_@@_siunitx_loaded_bool }
      { }
  }
```

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

```
\renewcommand*{\NC@rewrite@S}[1][]
  {
    \@temptokena \exp_after:wN
      {
        \tex_the:D \@temptokena
        > { \__siunitx_table_collect_begin: S {#1} }
        c
        < { \__siunitx_table_print: }
      }
    \NC@find
  }
```

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

```
\renewcommand*{\NC@rewrite@S}[1][]
  {
    \@temptokena \exp_after:wN
      {
        \tex_the:D \@temptokena
        > { \@@_Cell: \__siunitx_table_collect_begin: S {#1} }
        c
        < { \__siunitx_table_print: \@@_end_Cell: }
      }
    \NC@find
  }
```

However, we don't want do use explicitly any private command of siunitx. That's why we will extract the name of the two `\__siunitx...` commands by their position in the code of `\NC@rewrite@S`. Since the command `\NC@rewrite@S` appends some tokens to the *toks* list `\@temptokena`, we use the LaTeX command `\NC@rewrite@S` in a group (`\group_begin:`–`\group_end:`) and we extract the two command names which are in the toks `\@temptokena`. However, this extraction can be done only when siunitx is loaded (and it may be loaded after nicematrix) and, in fact, after the beginning of the document — because some instructions of siunitx are executed in a `\AtBeginDocument`). That's why this extraction will be done only at the first use of an environment of nicematrix with the command `\@@_adapt_S_column:`.

```
\cs_set_protected:Npn \@@_adapt_S_column:
  {
    \bool_if:NT \c_@@_siunitx_loaded_bool
      {
        \group_begin:
        \@temptokena = { }
```

We protect `\NC@find` which is at the end of `\NC@rewrite@S`.

```
        \cs_set_eq:NN \NC@find \prg_do_nothing:
        \NC@rewrite@S { }
```

Conversion of the *toks* `\@temptokena` in a token list of expl3 (the toks are not supported by expl3 but we can, nevertheless, use the option `V` for `\tl_gset:NV`).

```
        \tl_gset:NV \g_tmpa_tl \@temptokena
        \group_end:
        \tl_new:N \c_@@_table_collect_begin_tl
        \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
        \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
        \tl_new:N \c_@@_table_print_tl
        \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
```

The token lists `\c_@@_table_collect_begin_tl` and `\c_@@_table_print_tl` contain now the two commands of siunitx.

If the adaptation has been done, the command `\@@_adapt_S_column:` becomes no-op (globally).

```
        \cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
      }
  }
```

The command `\@@_renew_NC@rewrite@S:` will be used in each environment of nicematrix in order to "rewrite" the `S` column in each environment (only if the boolean `\c_@@_siunitx_loaded_bool` is raised, of course).

```
\cs_new_protected:Npn \@@_renew_NC@rewrite@S:
  {
    \renewcommand*{\NC@rewrite@S}[1][]
      {
        \@temptokena \exp_after:wN
          {
            \tex_the:D \@temptokena
            > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} }
            c
            < { \c_@@_table_print_tl \@@_end_Cell: }
          }
        \NC@find
      }
  }
```

Parameters

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

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

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

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

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

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

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

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

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

```
\int_new:N \g_@@_NiceMatrixBlock_int
```

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

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

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`.

```
\seq_new:N \g_@@_names_seq
```

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

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

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

```
\bool_new:N \l_@@_NiceArray_bool
```

If the user uses `{NiceTabular}`, we will raise the following flag.

```
\bool_new:N \l_@@_NiceTabular_bool
```

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

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

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

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

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

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

```
\str_new:N \g_@@_com_or_env_str
\str_set:Nn \g_@@_com_or_env_str { environment }
```

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

```
\cs_new:Npn \@@_full_name_env:
  {
    \str_if_eq:VnTF \g_@@_com_or_env_str { command }
      { command \space \c_backslash_str \g_@@_name_env_str }
      { environment \space \{ \g_@@_name_env_str \} }
  }
```

The following token list corresponds to the option `code-after` (it's also possible to set the value of that parameter with the command `\CodeAfter`).

```
\tl_new:N \g_@@_code_after_tl
```

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

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

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

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

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

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

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

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

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

The following flag will be raised when the key `code-before` is used in the environment. Indeed, if there is a `code-before` in the environment, we will manage to have the `row` nodes and the `col` nodes available *before* the creation of the array.

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

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

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

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

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

Some cells will be declared as "empty" (for example a cell with the instrution `\Cdot`).

```
\bool_new:N \g_@@_empty_cell_bool
```

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

```
\dim_new:N \@@_old_arraycolsep_dim
```

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

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

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

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

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

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

The sequence `\g_@@_pos_of_blocks_seq` will be used by the test of non-overlapping of two blocks and when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the blocks).

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

```
\seq_new:N \g_@@_pos_of_xdots_seq
```

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

Variables for the exterior rows and columns

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

- **First row**

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

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

- **First column**

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

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

- **Last row**

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

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

  If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the "last row".[30]

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

[30] We can't use `\l_@@_last_row_int` for this usage because, if nicematrix has read its value from the `aux` file, the value of the counter won't be -1 any longer.

Idem for `\l_@@_last_col_without_value_bool`

```
\bool_new:N \l_@@_last_col_without_value_bool
```

- **Last column**

  For the potential "last column", we use an integer. A value of -2 means that there is no last column. A value of -1 means that there is a last column but we don't know its value because the user has used the option `last-col` without value (it's possible in an environment without preamble like `{pNiceMatrix}`). A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`).

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

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

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

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

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

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

Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).
`#1` is the name of the node which will be created; `#2` and `#3` are the coordinates of one of the corner of the rectangle; `#4` and `#5` are the coordinates of the opposite corner.

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

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

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

The options

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.

```
\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).

```
\dim_new:N \l_@@_cell_space_top_limit_dim
\dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
\dim_new:N \l_@@_inter_dots_dim
\dim_set:Nn \l_@@_inter_dots_dim { 0.45 em }
```

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

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

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

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

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

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

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

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

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

```
\str_new:N \l_@@_baseline_str
\str_set:Nn \l_@@_baseline_str c
```

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

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

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

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

The flag `\l_@@_hlines_bool` corresponds to the key `hlines`, the flag `\l_@@_vlines_bool` to the key `vlines` and the flag `hvlines` to the key `hvlines`. Since version 4.1, the key `hvlines` is no longer a mere alias for the conjonction of `hlines` and `vlines`. Indeed, with `hvlines`, the vertical and horizontal rules are *not* drawn within the blocks (created by `\Block`).

```
\bool_new:N \l_@@_hlines_bool
\bool_new:N \l_@@_vlines_bool
\bool_new:N \l_@@_hvlines_bool
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To acheive 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 max-delimiter-width which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

```
\bool_new:N \l_@@_max_delimiter_width_bool
```

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

```
\keys_define:nn { NiceMatrix / xdots }
  {
    line-style .code:n =
     {
       \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.

```
         { \cs_if_exist_p:N \tikzpicture }
         { \str_if_eq_p:nn { #1 } { standard } }
         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
         { \@@_error:n { bad~option~for~line-style } }
     } ,
    line-style .value_required:n = true ,
    color .tl_set:N = \l_@@_xdots_color_tl ,
    color .value_required:n = true ,
    shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
    shorten .value_required:n = true ,
```

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

```
    down .tl_set:N = \l_@@_xdots_down_tl ,
    up .tl_set:N = \l_@@_xdots_up_tl ,
    unknown .code:n = \@@_error:n { Unknown~option~for~xdots }
  }
```

```
\keys_define:nn { NiceMatrix / rules }
  {
    color .tl_set:N = \l_@@_rules_color_tl ,
    color .value_required:n = true ,
    width .dim_set:N = \arrayrulewidth ,
    width .value_required:n = true
  }
\keys_define:nn { NiceMatrix / Global }
  {
    standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
    standard-cline .default:n = true ,
    cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
    cell-space-top-limit .value_required:n = true ,
    cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
    cell-space-bottom-limit .value_required:n = true ,
    xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
    max-delimiter-width .bool_set:N = \l_@@_max_delimiter_width_bool ,
    light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
    light-syntax .default:n = true ,
    end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
    end-of-row .value_required:n = true ,
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    last-row .int_set:N = \l_@@_last_row_int ,
    last-row .default:n = -1 ,
    code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
    code-for-first-col .value_required:n = true ,
    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
    code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
    code-for-last-row .value_required:n = true ,
    hlines .bool_set:N = \l_@@_hlines_bool ,
    vlines .bool_set:N = \l_@@_vlines_bool ,
    hvlines .code:n =
     {
       \bool_set_true:N \l_@@_hvlines_bool
       \bool_set_true:N \l_@@_vlines_bool
       \bool_set_true:N \l_@@_hlines_bool
     } ,
    parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
```

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

```
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
    create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
    create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
    create-extra-nodes .meta:n =
      { create-medium-nodes , create-large-nodes } ,
    left-margin .dim_set:N = \l_@@_left_margin_dim ,
    left-margin .default:n = \arraycolsep ,
    right-margin .dim_set:N = \l_@@_right_margin_dim ,
    right-margin .default:n = \arraycolsep ,
    margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
    margin .default:n = \arraycolsep ,
    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
    extra-margin .meta:n =
      { extra-left-margin = #1 , extra-right-margin = #1 } ,
```

```
    extra-margin .value_required:n = true
  }
```

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

```
\keys_define:nn { NiceMatrix / Env }
  {
    code-before .code:n =
     {
       \tl_if_empty:nF { #1 }
         {
           \tl_set:Nn \l_@@_code_before_tl { #1 }
           \bool_set_true:N \l_@@_code_before_bool
         }
     } ,
```

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

```
    c .code:n = \str_set:Nn \l_@@_baseline_str c ,
    t .code:n = \str_set:Nn \l_@@_baseline_str t ,
    b .code:n = \str_set:Nn \l_@@_baseline_str b ,
    baseline .tl_set:N = \l_@@_baseline_str ,
    baseline .value_required:n = true ,
    columns-width .code:n =
      \str_if_eq:nnTF { #1 } { auto }
        { \bool_set_true:N \l_@@_auto_columns_width_bool }
        { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
    columns-width .value_required:n = true ,
    name .code:n =
```

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

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

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

```
\keys_define:nn { NiceMatrix }
  {
    NiceMatrixOptions .inherit:n =
      {
        NiceMatrix / Global ,
      } ,
    NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
    NiceTabular .inherit:n =
```

```
      {
        NiceMatrix / Global ,
        NiceMatrix / Env
      } ,
    NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
    NiceTabular / rules .inherit:n = NiceMatrix / rules ,
    NiceArray .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    NiceArray / rules .inherit:n = NiceMatrix / rules ,
    pNiceArray .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      } ,
    pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    pNiceArray / rules .inherit:n = NiceMatrix / rules ,
  }
```

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

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

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

```
    renew-matrix .code:n = \@@_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    transparent .meta:n = { renew-dots , renew-matrix } ,
    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}`.

```
    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.

```
    columns-width .code:n =
      \str_if_eq:nnTF { #1 } { auto }
        { \@@_error:n { Option~auto~for~columns-width } }
        { \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.

```
    allow-duplicate-names .code:n =
      \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
    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).

```
    letter-for-dotted-lines .code:n =
      {
        \int_compare:nTF { \tl_count:n { #1 } = 1 }
          { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
          { \@@_error:n { Bad~value~for~letter~for~dotted~lines } }
      } ,
    letter-for-dotted-lines .value_required:n = true ,
    unknown .code:n  = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
  }
\str_new:N \l_@@_letter_for_dotted_lines_str
\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.

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

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

```
\keys_define:nn { NiceMatrix / NiceMatrix }
  {
    last-col .code:n = \tl_if_empty:nTF {#1}
                         {
                           \bool_set_true:N \l_@@_last_col_without_value_bool
                           \int_set:Nn \l_@@_last_col_int { -1 }
                         }
                         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
    l .code:n = \tl_set:Nn \l_@@_type_of_col_tl L ,
    r .code:n = \tl_set:Nn \l_@@_type_of_col_tl R ,
    L .code:n = \tl_set:Nn \l_@@_type_of_col_tl L ,
    R .code:n = \tl_set:Nn \l_@@_type_of_col_tl R ,
    S .code:n = \bool_if:NTF \c_@@_siunitx_loaded_bool
                  { \tl_set:Nn \l_@@_type_of_col_tl S }
                  { \@@_error:n { option~S~without~siunitx } } ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
  }
```

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

```
\keys_define:nn { NiceMatrix / NiceArray }
  {
```

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.

```
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    last-col .code:n = \tl_if_empty:nF { #1 }
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
  }
\keys_define:nn { NiceMatrix / pNiceArray }
  {
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
```

```
                      \int_zero:N \l_@@_last_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix }
  }
```

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

```
\keys_define:nn { NiceMatrix / NiceTabular }
  {
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceTabular }
  }
```

Important code used by {NiceArrayWithDelims}

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

```
\cs_new_protected:Npn \@@_Cell:
  {
```

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

```
    \int_gincr:N \c@jCol
```

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

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

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

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

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).
The codes `\l_@@_code_for_first_row_tl` and *al* don't apply in the corners of the matrix.

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

```
            \xglobal \colorlet { nicematrix-last-row } { . }
          }
      }
  }
```

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

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

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines will be dynamically added to this command.

```
\cs_new_protected:Npn \@@_update_for_first_and_last_row:
  {
    \int_compare:nNnTF \c@iRow = 0
      {
        \dim_gset:Nn \g_@@_dp_row_zero_dim
          { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
        \dim_gset:Nn \g_@@_ht_row_zero_dim
          { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
      }
      {
        \int_compare:nNnT \c@iRow = 1
          {
            \dim_gset:Nn \g_@@_ht_row_one_dim
              { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
          }
      }
  }
\cs_new_protected:Npn \@@_end_Cell:
  {
    \@@_math_toggle_token:
    \hbox_set_end:
    \box_set_ht:Nn \l_@@_cell_box
      { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
    \box_set_dp:Nn \l_@@_cell_box
      { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the "first column" and the "last column").

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

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

```
    \@@_update_for_first_and_last_row:
```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

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

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

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

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

The first argument of the following command `\@@_instruction_of_type:nn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The second 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}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \cdot \end{pmatrix}$$

the content of `\g_@@_Cdots_lines_tl` will be:

```
\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{color=red}
```

```
\cs_new_protected:Npn \@@_instruction_of_type:nn #1 #2
  {
```

It's important to use a `\tl_gput_right:cx` and not a `\tl_gput_left:cx` because we want the `\Ddots` lines to be drawn in the order of appearance in the array (for parallelisation).

```
    \tl_gput_right:cx
      { g_@@_ #1 _ lines _ tl }
      {
        \use:c { @@ _ draw _ #1 : nnn }
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \exp_not:n { #2 } }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_array:
  {
    \bool_if:NTF \c_@@_revtex_bool
      {
        \cs_set_eq:NN \@acoll \@arrayacol
        \cs_set_eq:NN \@acolr \@arrayacol
        \cs_set_eq:NN \@acol \@arrayacol
        \cs_set:Npn \@halignto { }
        \@array@array
      }
      \array
```

`\l_@@_baseline_str` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of array) with the option `t` and the right translation will be done further.

```
    [ \str_if_eq:VnTF \l_@@_baseline_str c c t ]
  }
```

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

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

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

```
\cs_new_protected:Npn \@@_create_row_node:
  {
```

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

```
    \hbox
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \vtop
```

```
              {
                \skip_vertical:N 0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
                \skip_vertical:N -0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
          { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
        \str_if_empty:NF \l_@@_name_str
          {
            \pgfnodealias
              { \l_@@_name_str - row - \int_use:N \c@iRow }
              { \@@_env: - row - \int_use:N \c@iRow }
          }
        \endpgfpicture
      }
  }
```

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

```
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i:
  {
    \int_gzero:N \c@jCol
    \bool_if:NF \g_@@_row_of_col_done_bool
      {
        \@@_create_row_node:
```

We don't draw the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for theses rules.

```
        \bool_if:NT \l_@@_hlines_bool
          {
```

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.

```
            \int_compare:nNnT \c@iRow > { -1 }
              {
                \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@`.

```
                  { \hrule height \arrayrulewidth width \c_zero_dim }
              }
          }
      }
  }
```

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`, `W`, `p`, `m` and `b`).

```
\cs_set_protected:Npn \@@_newcolumntype #1
  {
    \cs_if_free:cT { NC @ find @ #1 }
      { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
    \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
    \peek_meaning:NTF [
      { \newcol@ #1 }
      { \newcol@ #1 [ 0 ] }
  }
```

The following command will be used to redefine the column types `p`, `m` and `b`. That means that it will be used three times. The first argument is the letter of the column type (`p`, `m` or `b`). The second is the letter of position for the environment `{minipage}` (`t`, `c` or `b`).

```
\cs_new_protected:Npn \@@_define_columntype:nn #1 #2
  {
```

We don't want a warning for redefinition of the column type. That's why we use `\@@_newcolumntype` and not `\newcolumntype`.

```
    \@@_newcolumntype #1 [ 1 ]
      {
        > {
            \@@_Cell:
            \begin { minipage } [ #2 ] { ##1 }
            \mode_leave_vertical: \box_use:N \@arstrutbox
          }
```

Here, we put `c` but we would have the result with `l` or `r`.

```
        c
        < { \box_use:N \@arstrutbox \end { minipage } \@@_end_Cell: }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_pre_array:
  {
```

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 [31].

```
    \bool_if:NT \c_@@_booktabs_loaded_bool
      { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
    \box_clear_new:N \l_@@_cell_box
    \cs_if_exist:NT \theiRow
      { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
    \int_gzero_new:N \c@iRow
    \cs_if_exist:NT \thejCol
      { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
    \int_gzero_new:N \c@jCol
    \normalbaselines
```

If 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}`).

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

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

```
    \cs_set:Npn \ialign
      {
        \bool_if:NT \l_@@_NiceTabular_bool
```

[31] cf. `\nicematrix@redefine@check@rerun`

```
      { \dim_set_eq:NN \arraycolsep \@@_old_arraycolsep_dim }
    \bool_if:NTF \c_@@_colortbl_loaded_bool
      {
        \CT@everycr
          {
            \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
            \@@_everycr:
          }
      }
      { \everycr { \@@_everycr: } }
    \tabskip = \c_zero_skip
```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current values of `\arraystretch`[32] 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`.

```
    \dim_gzero_new:N \g_@@_dp_row_zero_dim
    \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
    \dim_gzero_new:N \g_@@_ht_row_zero_dim
    \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
    \dim_gzero_new:N \g_@@_ht_row_one_dim
    \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
    \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
    \dim_gzero_new:N \g_@@_ht_last_row_dim
    \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
    \dim_gzero_new:N \g_@@_dp_last_row_dim
    \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 programmation, we will have, in the cells of the array, a clean version of `\ialign`.[33]

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

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

```
\cs_set_eq:NN \@@_old_ldots \ldots
\cs_set_eq:NN \@@_old_cdots \cdots
\cs_set_eq:NN \@@_old_vdots \vdots
\cs_set_eq:NN \@@_old_ddots \ddots
\cs_set_eq:NN \@@_old_iddots \iddots
\cs_set_eq:NN \firsthline \hline
\cs_set_eq:NN \lasthline \hline
\bool_if:NTF \l_@@_standard_cline_bool
  { \cs_set_eq:NN \cline \@@_standard_cline }
  { \cs_set_eq:NN \cline \@@_cline }
\cs_set_eq:NN \Ldots \@@_Ldots
\cs_set_eq:NN \Cdots \@@_Cdots
\cs_set_eq:NN \Vdots \@@_Vdots
\cs_set_eq:NN \Ddots \@@_Ddots
\cs_set_eq:NN \Iddots \@@_Iddots
\cs_set_eq:NN \hdottedline \@@_hdottedline:
\cs_set_eq:NN \Hspace \@@_Hspace:
\cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
\cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
\cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
\cs_set_eq:NN \Block \@@_Block:
```

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

[33] The user will probably not use directly `\ialign` in the array... but more likely environments that utilize `\ialign` internally (e.g.: `{substack}`).

```
    \cs_set_eq:NN \rotate \@@_rotate:
    \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
    \cs_set_eq:NN \dotfill \@@_dotfill:
    \cs_set_eq:NN \CodeAfter \@@_CodeAfter:n
    \cs_set_eq:NN \diagbox \@@_diagbox:nn
    \bool_if:NT \l_@@_renew_dots_bool
      {
        \cs_set_eq:NN \ldots \@@_Ldots
        \cs_set_eq:NN \cdots \@@_Cdots
        \cs_set_eq:NN \vdots \@@_Vdots
        \cs_set_eq:NN \ddots \@@_Ddots
        \cs_set_eq:NN \iddots \@@_Iddots
        \cs_set_eq:NN \dots \@@_Ldots
        \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
      }
```

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).

```
    \seq_gclear_new:N \g_@@_multicolumn_cells_seq
    \seq_gclear_new: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).

```
    \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`).

```
    \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.

```
    \int_gzero_new:N \g_@@_col_total_int
    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

We define the new column types `L`, `C` and `R` that must be used instead of `l`, `c` and `r` in the preamble of `{NiceArray}`. We use `\@@_newcolumntype` because it will be slightly quicker thant `\newcolumtype`.

```
    \@@_newcolumntype L { > \@@_Cell: l < \@@_end_Cell: }
    \@@_newcolumntype C { > \@@_Cell: c < \@@_end_Cell: }
    \@@_newcolumntype R { > \@@_Cell: r < \@@_end_Cell: }
```

We redefine the column types `p`, `m` and `b`. The command `\@@_define_columntype:nn` is only used here.

```
    \@@_define_columntype:nn p t
    \@@_define_columntype:nn m c
    \@@_define_columntype:nn b b
```

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumtype` because we don't want warnings for column types already defined.

```
    \@@_newcolumntype w [ 2 ]
      {
        > {
            \hbox_set:Nw \l_@@_cell_box
            \@@_Cell:
          }
        c
        < {
            \@@_end_Cell:
            \hbox_set_end:
```

The `\str_lowercase:n` is only for giving the user the ability to write `wC{1cm}` instead of `wc{1cm}` for homogeneity with the letters `L`, `C` and `R` used elsewhere in the preamble instead of `l`, `c` and `r`.

```
            \makebox [ ##2 ] [ \str_lowercase:n { ##1 } ]
               { \box_use_drop:N \l_@@_cell_box }
          }
      }
    \@@_newcolumntype W [ 2 ]
      {
        > {
            \hbox_set:Nw \l_@@_cell_box
            \@@_Cell:
          }
        c
        < {
            \@@_end_Cell:
            \hbox_set_end:
            \cs_set_eq:NN \hss \hfil
            \makebox [ ##2 ] [ \str_lowercase:n { ##1 } ]
              { \box_use_drop:N \l_@@_cell_box }
          }
      }
```

By default, the letter used to specify a dotted line in the preamble of an environment of nicematrix (for example in `{pNiceArray}`) is the letter `:`. However, this letter is used by some packages, for example arydshln. That's why it's possible to change the letter used by nicematrix with the option `letter-for-dotted-lines` which changes the value of `\l_@@_letter_for_dotted_lines_str`. We rescan this string (which is always of length 1) in particular for the case where `pdflatex` is used with french-babel (the colon is activated by french-babel at the beginning of the document).

```
    \tl_set_rescan:Nno
      \l_@@_letter_for_dotted_lines_str { } \l_@@_letter_for_dotted_lines_str
    \exp_args:NV \newcolumntype \l_@@_letter_for_dotted_lines_str
      {
        !
          {
```

The following code 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).

```
            \int_compare:nNnF \c@iRow = 0
              {
                \int_compare:nNnF \c@iRow = \l_@@_last_row_int
                  { \skip_horizontal:N 2\l_@@_radius_dim }
              }
```

Consider the following code:

```
\begin{NiceArray}{C:CC:C}
a & b
c & d \\
e & f & g & h \\
i & j & k & l
\end{NiceArray}
```

The first "`:`" in the preamble will be encountered during the first row of the environment `{NiceArray}` but the second one will be encountered only in the third row. We have to issue a command `\vdottedline:n` in the `code-after` only one time for each "`:`" in the preamble. That's why we keep a counter `\g_@@_last_vdotted_col_int` and with this counter, we know whether a letter "`:`" encountered during the parsing has already been taken into account in the `code-after`.

```
            \int_compare:nNnT \c@jCol > \g_@@_last_vdotted_col_int
              {
                \int_gset_eq:NN \g_@@_last_vdotted_col_int \c@jCol
                \tl_gput_right:Nx \g_@@_internal_code_after_tl
```

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

```
                    { \@@_vdottedline:n { \int_use:N \c@jCol } }
                }
            }
        }
      \int_gzero_new:N \g_@@_last_vdotted_col_int
      \bool_if:NT \c_@@_siunitx_loaded_bool \@@_renew_NC@rewrite@S:
      \int_gset:Nn \g_@@_last_vdotted_col_int { -1 }
      \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.

```
      \tl_gclear_new:N \g_@@_Cdots_lines_tl
      \tl_gclear_new:N \g_@@_Ldots_lines_tl
      \tl_gclear_new:N \g_@@_Vdots_lines_tl
      \tl_gclear_new:N \g_@@_Ddots_lines_tl
      \tl_gclear_new:N \g_@@_Iddots_lines_tl
      \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl
  }
```

The environment {NiceArrayWithDelims}

```
\NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } }
  {
    \tl_set:Nn \l_@@_left_delim_tl { #1 }
    \tl_set:Nn \l_@@_right_delim_tl { #2 }
    \bool_gset_false:N \g_@@_row_of_col_done_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
    \@@_adapt_S_column:
    \bool_if:NTF \l_@@_NiceTabular_bool
      \mode_leave_vertical:
      \@@_test_if_math_mode:
    \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
    \bool_set_true:N \l_@@_in_env_bool
```

The command \CT@arc@ contains the instruction of color for the rules of the array[34]. 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.

```
      \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).

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

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

```
    \int_gincr:N \g_@@_env_int
    \bool_if:NF \l_@@_block_auto_columns_width_bool
      { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

We do a redefinition of \@arrayrule because we want that the vertical rules drawn by | in the preamble of the array don't extend in the potential exterior rows.

```
    \cs_set_protected:Npn \@arrayrule { \@addtopreamble \@@_vline: }
```

[34] e.g. \color[rgb]{0.5,0.5,0})

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.

```
\seq_clear:N \g_@@_blocks_seq
\seq_clear:N \g_@@_pos_of_blocks_seq
```

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

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


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

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

```
\bool_if:NT \l_@@_code_before_bool
  {
    \seq_if_exist:cT { @@_size_ \int_use:N \g_@@_env_int _ seq }
      {
```

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

```
        \int_zero_new:N \c@iRow
        \int_set:Nn \c@iRow
          { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 2 }
        \int_zero_new:N \c@jCol
        \int_set:Nn \c@jCol
          { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 }
```

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

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

Now, we will create all the 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.

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

First, the creation of the row nodes.

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

Now, the creation of the col nodes.

```
        \int_step_inline:nnn
          { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 3 }
          { \seq_item:cn { @@_size_ \int_use:N \g_@@_env_int _ seq } 4 + 1 }
          {
            \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
            \pgfcoordinate { \@@_env: - col - ##1 }
```

```
                { \pgfpointdiff \@@_picture_position: \@@_node_position: }
            }
          \endpgfpicture
          \group_begin:
            \bool_if:NT \c_@@_tikz_loaded_bool
              {
                \tikzset
                  {
                    every~picture / .style =
                      { overlay , name~prefix = \@@_env: - }
                  }
              }
            \cs_set_eq:NN \cellcolor \@@_cellcolor
            \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
            \cs_set_eq:NN \rowcolor \@@_rowcolor
            \cs_set_eq:NN \rowcolors \@@_rowcolors
            \cs_set_eq:NN \columncolor \@@_columncolor
            \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
```

We compose the `code-before` in math mode in order to nullify the spaces put by the user between instructions in the `code-before`.

```
            \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
            \l_@@_code_before_tl
            \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
          \group_end:
        }
    }
```

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

```
    \int_compare:nNnT \l_@@_last_row_int > { -2 }
      {
        \tl_put_right:Nn \@@_update_for_first_and_last_row:
          {
            \dim_gset:Nn \g_@@_ht_last_row_dim
              { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
            \dim_gset:Nn \g_@@_dp_last_row_dim
              { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
          }
      }
    \int_compare:nNnT \l_@@_last_row_int = { -1 }
      {
        \bool_set_true:N \l_@@_last_row_without_value_bool
```

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

```
        \str_if_empty:NTF \l_@@_name_str
          {
            \cs_if_exist:cT { @@_last_row_ \int_use:N \g_@@_env_int }
              {
                \int_set:Nn \l_@@_last_row_int
                  { \use:c { @@_last_row_ \int_use:N \g_@@_env_int } }
              }
          }
          {
            \cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
              {
                \int_set:Nn \l_@@_last_row_int
                  { \use:c { @@_last_row_ \l_@@_name_str } }
              }
          }
      }
```

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

```
    \int_compare:nNnT \l_@@_last_col_int = { -1 }
      {
        \str_if_empty:NTF \l_@@_name_str
          {
            \cs_if_exist:cT { @@_last_col_ \int_use:N \g_@@_env_int }
              {
                \int_set:Nn \l_@@_last_col_int
                  { \use:c { @@_last_col_ \int_use:N \g_@@_env_int } }
              }
          }
          {
            \cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
              {
                \int_set:Nn \l_@@_last_col_int
                  { \use:c { @@_last_col_ \l_@@_name_str } }
              }
          }
      }
```

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

```
    \@@_pre_array:
```

We compute the width of the two delimiters.

```
    \dim_zero_new:N \l_@@_left_delim_dim
    \dim_zero_new:N \l_@@_right_delim_dim
    \bool_if:NTF \l_@@_NiceArray_bool
      {
        \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
        \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
      }
      {
```

The command `\bBigg@` is a command of amsmath.

```
        \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 #1 $ }
        \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
        \hbox_set:Nn \l_tmpa_box { $\bBigg@ 5 #2 $ }
        \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
      }
```

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

```
    \box_clear_new:N \l_@@_the_array_box
```

We construct the preamble of the array in `\l_tmpa_tl`.

```
    \tl_set:Nn \l_tmpa_tl { #4 }
    \int_compare:nNnTF \l_@@_first_col_int = 0
      { \tl_put_left:NV \l_tmpa_tl \c_@@_preamble_first_col_tl }
      {
        \bool_lazy_all:nT
          {
            \l_@@_NiceArray_bool
            { \bool_not_p:n \l_@@_NiceTabular_bool }
            { \bool_not_p:n \l_@@_vlines_bool }
            { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
          }
          { \tl_put_left:Nn \l_tmpa_tl { @ { } } }
      }
    \int_compare:nNnTF \l_@@_last_col_int > { -1 }
      { \tl_put_right:NV \l_tmpa_tl \c_@@_preamble_last_col_tl }
      {
        \bool_lazy_all:nT
          {
            \l_@@_NiceArray_bool
```

```
            { \bool_not_p:n \l_@@_NiceTabular_bool }
            { \bool_not_p:n \l_@@_vlines_bool }
            { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
          }
          { \tl_put_right:Nn \l_tmpa_tl { @ { } } }
      }
    \tl_put_right:Nn \l_tmpa_tl { > { \@@_error_too_much_cols: } L }
```

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

```
    \hbox_set:Nw \l_@@_the_array_box
```

Here is a trick. We will call `\array` and, at the beginning, `\array` will set `\col@sep` equal to the current value of `\arraycolsep`. In we are in an environment `{NiceTabular}`, we would like that `\array` sets `\col@sep` equal to the current value of `\tabcolsep`. That's why we set `\arraycolsep` equal to `\tabcolsep`. However, the value of `\tabcolsep` in each cell of the array should be equal to the current value of `\tabcolsep` outside `{NiceTabular}`. That's why we save the current value of `\arraycolsep` and we will restore the value just before the `\halign`. It's possible because we do a redefinition of `\ialign` (see just below).

```
    \bool_if:NT \l_@@_NiceTabular_bool
      {
        \dim_set_eq:NN \@@_old_arraycolsep_dim \arraycolsep
        \dim_set_eq:NN \arraycolsep \tabcolsep
      }
```

If the key `\vlines` is used, we increase `\arraycolsep` by `0.5\arrayrulewidth` in order to reserve space for the width of the vertical rules drawn with Tikz after the end of the array. However, the first `\arraycolsep` is used once (between columns, `\arraycolsep` is used twice). That's why we add a `0.5\arrayrulewidth` more.

```
    \bool_if:NT \l_@@_vlines_bool
      {
        \dim_add:Nn \arraycolsep { 0.5 \arrayrulewidth }
        \skip_horizontal:N  0.5\arrayrulewidth
      }
    \skip_horizontal:N \l_@@_left_margin_dim
    \skip_horizontal:N \l_@@_extra_left_margin_dim
    \c_math_toggle_token
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { @@-light-syntax } }
      { \use:c { @@-normal-syntax } }
  }
  {
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { end @@-light-syntax } }
      { \use:c { end @@-normal-syntax } }
    \c_math_toggle_token
    \skip_horizontal:N \l_@@_right_margin_dim
    \skip_horizontal:N \l_@@_extra_right_margin_dim
```

If the key `\vlines` is used, we have increased `\arraycolsep` by `0.5\arrayrulewidth` in order to reserve space for the width of the vertical rules drawn with Tikz after the end of the array. However, the last `\arraycolsep` is used once (between columns, `\arraycolsep` is used twice). That's we add a `0.5 \arrayrulewidth` more.

```
    \bool_if:NT \l_@@_vlines_bool { \skip_horizontal:N 0.5\arrayrulewidth }
    \hbox_set_end:
```

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

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

```
    \int_compare:nNnT \l_@@_last_row_int > { -2 }
      {
        \bool_if:NF \l_@@_last_row_without_value_bool
```

```
        {
          \int_compare:nNnF \l_@@_last_row_int = \c@iRow
            {
              \@@_error:n { Wrong~last~row }
              \int_gset_eq:NN \l_@@_last_row_int \c@iRow
            }
        }
    }
```

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".[35]

```
    \int_gset_eq:NN \c@jCol \g_@@_col_total_int
    \bool_if:nT \g_@@_last_col_found_bool { \int_gdecr:N \c@jCol }
```

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

```
    \int_gset_eq:NN \g_@@_row_total_int \c@iRow
    \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. 70).

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

The construction of the real box is different in `{NiceArray}` and in the other environments because, in `{NiceArray}`, we have to take into account the value of `baseline` and we have no delimiter to put. We begin with `{NiceArray}`.

```
    \bool_if:NTF \l_@@_NiceArray_bool
      {
```

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`.

```
        \str_if_eq:VnTF \l_@@_baseline_str { b }
          {
           \pgfpicture
             \@@_qpoint:n { row - 1 }
             \dim_gset_eq:NN \g_tmpa_dim \pgf@y
             \@@_qpoint:n { row - \int_use:N \c@iRow - base }
             \dim_gsub:Nn \g_tmpa_dim \pgf@y
           \endpgfpicture
           \int_compare:nNnT \l_@@_first_row_int = 0
             {
               \dim_gadd:Nn \g_tmpa_dim
                 { \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
             }
           \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_@@_the_array_box }
          }
          {
            \str_if_eq:VnTF \l_@@_baseline_str { c }
              { \box_use_drop:N \l_@@_the_array_box }
              {
```

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

```
                \str_if_eq:VnT \l_@@_baseline_str { t }
                  { \str_set:Nn \l_@@_baseline_str { 1 } }
```

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

```
                \int_set:Nn \l_tmpa_int \l_@@_baseline_str
                \bool_if:nT
                  {
```

[35] We remind that the potential "first column" (exterior) has the number 0.

```
                \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int
              || \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int
            }
            {
              \@@_error:n { bad~value~for~baseline }
              \int_set:Nn \l_tmpa_int 1
            }
          \pgfpicture
          \@@_qpoint:n { row - 1 }
          \dim_gset_eq:NN \g_tmpa_dim \pgf@y
          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
          \dim_gsub:Nn \g_tmpa_dim \pgf@y
          \endpgfpicture
          \int_compare:nNnT \l_@@_first_row_int = 0
            {
              \dim_gadd:Nn \g_tmpa_dim
                { \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
            }
          \box_move_up:nn \g_tmpa_dim
            { \box_use_drop:N \l_@@_the_array_box }
        }
    }
}
```

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).

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

We compute `\l_tmpb_dim` which is the total height of the "last row" below the array (when the key `last-row` is used). A value of -2 for `\l_@@_last_row_int` means that there is no "last row".[36]

```
  \int_compare:nNnTF \l_@@_last_row_int > { -2 }
    {
      \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
      \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
    }
    { \dim_zero:N \l_tmpb_dim }
  \hbox_set:Nn \l_tmpa_box
    {
      \c_math_toggle_token
      \left #1
      \vcenter
        {
```

We take into account the "first row" (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```
          \skip_vertical:N -\l_tmpa_dim
          \hbox
            {
              \bool_if:NTF \l_@@_NiceTabular_bool
                { \skip_horizontal:N -\tabcolsep }
                { \skip_horizontal:N -\arraycolsep }
              \box_use_drop:N \l_@@_the_array_box
              \bool_if:NTF \l_@@_NiceTabular_bool
                { \skip_horizontal:N -\tabcolsep }
                { \skip_horizontal:N -\arraycolsep }
            }
```

[36] A 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).

We take into account the “last row” (we have previously computed its total height in \l_tmpb_dim).

```
                \skip_vertical:N -\l_tmpb_dim
              }
            \right #2
            \c_math_toggle_token
          }
```

Now, the box \l_tmpa_box is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option max-delimiter-width is used.

```
        \bool_if:NTF \l_@@_max_delimiter_width_bool
          { \@@_put_box_in_flow_bis:nn { #1 } { #2 } }
          \@@_put_box_in_flow:
      }
```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in \g_@@_width_last_col_dim: see p. 71).

```
    \bool_if:NT \g_@@_last_col_found_bool
      {
        \skip_horizontal:N \g_@@_width_last_col_dim
        \skip_horizontal:N \arraycolsep
      }
    \@@_after_array:
  }
```

This is the end of the environment {NiceArrayWithDelims}.

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).

```
\cs_new_protected:Npn \@@_put_box_in_flow:
  {
    \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
    \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
    \str_if_eq:VnTF \l_@@_baseline_str { c }
      { \box_use_drop:N \l_tmpa_box }
      \@@_put_box_in_flow_i:
  }
```

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

```
\cs_new_protected:Npn \@@_put_box_in_flow_i:
  {
    \str_case:VnF \l_@@_baseline_str
      {
        { t } { \int_set:Nn \l_tmpa_int 1 }
        { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
      }
      { \int_set:Nn \l_tmpa_int \l_@@_baseline_str }
    \bool_if:nT
      {
           \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int
        || \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int
      }
      {
        \@@_error:n { bad~value~for~baseline }
        \int_set:Nn \l_tmpa_int 1
      }
    \pgfpicture
      \@@_qpoint:n { row - 1 }
      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \@@_succ:n \c@iRow }
      \dim_gadd:Nn \g_tmpa_dim \pgf@y
      \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }
```

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

```
      \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
      \dim_gsub:Nn \g_tmpa_dim \pgf@y
```

We take into account the position of the mathematical axis.

```
      \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
```

Now, \g_tmpa_dim contains the value of the y translation we have to to.

```
    \endpgfpicture
    \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
    \box_use_drop:N \l_tmpa_box
  }
```

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

```
\cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
  {
```

We will compute the real width of both delimiters used.

```
    \dim_zero_new:N \l_@@_real_left_delim_dim
    \dim_zero_new:N \l_@@_real_right_delim_dim
    \hbox_set:Nn \l_tmpb_box
      {
        \c_math_toggle_token
        \left #1
        \vcenter
          {
            \vbox_to_ht:nn
              { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
              { }
          }
        \right .
        \c_math_toggle_token
      }
    \dim_set:Nn \l_@@_real_left_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
    \hbox_set:Nn \l_tmpb_box
      {
        \c_math_toggle_token
        \left .
        \vbox_to_ht:nn
          { \box_ht:N \l_tmpa_box + \box_dp:N \l_tmpa_box }
          { }
        \right #2
        \c_math_toggle_token
      }
    \dim_set:Nn \l_@@_real_right_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
    \skip_horizontal:N  \l_@@_left_delim_dim
    \skip_horizontal:N -\l_@@_real_left_delim_dim
    \@@_put_box_in_flow:
    \skip_horizontal:N \l_@@_right_delim_dim
    \skip_horizontal:N -\l_@@_real_right_delim_dim
  }
```

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 used 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 arrray).

```
\NewDocumentEnvironment { @@-normal-syntax } { }
```

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

```
  {
    \peek_meaning_ignore_spaces:NTF \end
      { \@@_analyze_end:Nn }
```

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

```
      { \exp_args:NV \@@_array: \l_tmpa_tl }
  }
  {
    \@@_create_col_nodes:
    \endarray
  }
```

When the key `light-syntax` is used, we use an environment which takes its whole body as an argument (with the specifier `b` of xparse).

```
\NewDocumentEnvironment { @@-light-syntax } { b }
  {
```

First, we test whether the environment is empty. It's only a security. Of course, this test is more easy than the similar test for the "normal syntax" because we have the whole body of the environment in `#1`.

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

Now, you extract the `code-after` or the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be catched in the value of `\g_@@_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_@@_code_after_tl`.

```
    \@@_light_syntax_i #1 \CodeAfter \q_stop
  }
```

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

```
  { }
\cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
  {
    \tl_gput_right:Nn \g_@@_code_after_tl { #2 }
```

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

```
    \seq_gclear_new:N \g_@@_rows_seq
    \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
    \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

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

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

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

```
    \exp_args:NV \@@_array: \l_tmpa_tl
```

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

```
    \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
    \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
    \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
    \@@_create_col_nodes:
    \endarray
  }
\cs_new_protected:Npn \@@_line_with_light_syntax:n #1
  { \tl_if_empty:nF { #1 } { \\ \@@_line_with_light_syntax_i:n { #1 } } }
\cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
  {
    \seq_gclear_new:N \g_@@_cells_seq
    \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
    \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
    \l_tmpa_tl
    \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
  }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security).

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

We reput in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```
    \end { #2 }
  }
```

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

```
\cs_new:Npn \@@_create_col_nodes:
  {
    \crcr
    \int_compare:nNnT \c@iRow = 0 { \@@_fatal:n { Zero~row } }
    \int_compare:nNnT \l_@@_first_col_int = 0
      {
        \omit
        \skip_horizontal:N  -2\col@sep
        \bool_if:NT \l_@@_code_before_bool
          { \pgfsys@markposition { \@@_env: - col - 0 } }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
        \endpgfpicture
        &
      }
    \omit
```

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

```
    \bool_gset_true:N \g_@@_row_of_col_done_bool
```

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

```
    \int_compare:nNnTF \l_@@_first_col_int = 0
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
```

```
                \skip_horizontal:N -0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - 1 }
                \skip_horizontal:N 0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
        \endpgfpicture
      }
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:N 0.5 \arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - 1 }
                \skip_horizontal:N -0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
        \endpgfpicture
      }
```

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

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

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

```
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
        \endpgfpicture
      }
```

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.

```
    \int_gset:Nn \g_tmpa_int 1
    \bool_if:NTF \g_@@_last_col_found_bool
      { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
      { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
      {
        &
        \omit
```

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

```
        \int_gincr:N \g_tmpa_int
        \skip_horizontal:N \g_tmpa_skip
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:N -0.5\arrayrulewidth
                \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
                \skip_horizontal:N 0.5\arrayrulewidth
              }
          }
```

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

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

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

```
\tl_const:Nn \c_@@_preamble_first_col_tl
  {
    >
      {
        \@@_begin_of_row:
```

The contents of the cell is constructed in the box `\l_@@_cell_box` because we have to compute some dimensions of this box.

```
        \hbox_set:Nw \l_@@_cell_box
        \@@_math_toggle_token:
        \bool_if:NT \l_@@_small_bool \scriptstyle
```

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

```
        \bool_lazy_and:nnT
          { \int_compare_p:nNn \c@iRow > 0 }
          {
            \bool_lazy_or_p:nn
              { \int_compare_p:nNn \l_@@_last_row_int < 0 }
              { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
          }
          {
            \l_@@_code_for_first_col_tl
            \xglobal \colorlet { nicematrix-first-col } { . }
          }
      }
```

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

```
    l
    <
      {
        \@@_math_toggle_token:
        \hbox_set_end:
        \@@_update_for_first_and_last_row:
```

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

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

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

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

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

```
\tl_const:Nn \c_@@_preamble_last_col_tl
  {
    >
      {
```

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

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

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

```
        \hbox_set:Nw \l_@@_cell_box
          \@@_math_toggle_token:
          \bool_if:NT \l_@@_small_bool \scriptstyle
```

We insert `\l_@@_code_for_last_col_tl`... but we don't insert it in the potential "first row" and in the potential "last row".

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

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

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

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

```
        \hbox_overlap_right:n
          {
            \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
              {
                \skip_horizontal:N \l_@@_right_delim_dim
                \skip_horizontal:N \l_@@_right_margin_dim
                \skip_horizontal:N \l_@@_extra_right_margin_dim
                \@@_node_for_the_cell:
              }
          }
      }
  }
```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\l_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```
\NewDocumentEnvironment { NiceArray } { }
  {
    \bool_set_true:N \l_@@_NiceArray_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { NiceArray } }
```

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).

```
    \NiceArrayWithDelims . .
  }
  { \endNiceArrayWithDelims }
```

We create the variants of the environment `{NiceArrayWithDelims}`.

```
\NewDocumentEnvironment { pNiceArray } { }
```

```
  {
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { pNiceArray } }
    \@@_test_if_math_mode:
    \NiceArrayWithDelims ( )
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { bNiceArray } { }
  {
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { bNiceArray } }
    \@@_test_if_math_mode:
    \NiceArrayWithDelims [ ]
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { BNiceArray } { }
  {
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { BNiceArray } }
    \@@_test_if_math_mode:
    \NiceArrayWithDelims \{ \}
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { vNiceArray } { }
  {
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { vNiceArray } }
    \@@_test_if_math_mode:
    \NiceArrayWithDelims | |
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { VNiceArray } { }
  {
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { VNiceArray } }
    \@@_test_if_math_mode:
    \NiceArrayWithDelims \| \|
  }
  { \endNiceArrayWithDelims }
```

The environment {NiceMatrix} and its variants

```
\cs_new_protected:Npn \@@_define_env:n #1
  {
    \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
      {
        \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
        \tl_set:Nn \l_@@_type_of_col_tl C
        \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
        \exp_args:Nnx \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
      }
      { \use:c { end #1 NiceArray } }
  }
\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
  {
    \use:c { #1 NiceArray }
      {
        *
          {
            \int_compare:nNnTF \l_@@_last_col_int < 0
              \c@MaxMatrixCols
```

```
              { \@@_pred:n \l_@@_last_col_int }
          }
          #2
      }
  }
\@@_define_env:n { }
\@@_define_env:n p
\@@_define_env:n b
\@@_define_env:n B
\@@_define_env:n v
\@@_define_env:n V
```

The environment {NiceTabular}

```
\NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
  {
    \str_gset:Nn \g_@@_name_env_str { NiceTabular }
    \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
    \bool_set_true:N \l_@@_NiceTabular_bool
    \NiceArray { #2 }
  }
  { \endNiceArray }
```

After the construction of the array

```
\cs_new_protected:Npn \@@_after_array:
  {
    \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 `\Vdots` drawn in that last column. That's why we fix the correct value of `\l_@@_last_col_int` in that case.

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

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

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

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

```
    \bool_if:NT \l_@@_last_row_without_value_bool
```

```
    {
      \dim_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int
```

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

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

If the environment has a name, we also write a value based on the name because it's more reliable than a value based on the number of the environment.

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

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

```
  \bool_if:NT \l_@@_code_before_bool
    {
      \iow_now:Nn \@mainaux \ExplSyntaxOn
      \iow_now:Nx \@mainaux
        { \seq_clear_new:c { @@_size _ \int_use:N \g_@@_env_int _ seq } }
      \iow_now:Nx \@mainaux
        {
          \seq_gset_from_clist:cn { @@_size _ \int_use:N \g_@@_env_int _ seq }
            {
              \int_use:N \l_@@_first_row_int ,
              \int_use:N \g_@@_row_total_int ,
              \int_use:N \l_@@_first_col_int ,
```

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

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

By default, the diagonal lines will be parallelized[37]. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

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

[37] It's possible to use the option `parallelize-diags` to disable this parallelization.

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` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```
      \dim_gzero_new:N \g_@@_delta_x_one_dim
      \dim_gzero_new:N \g_@@_delta_y_one_dim
      \dim_gzero_new:N \g_@@_delta_x_two_dim
      \dim_gzero_new:N \g_@@_delta_y_two_dim
    }
  \bool_if:nTF \l_@@_medium_nodes_bool
    {
      \bool_if:NTF \l_@@_large_nodes_bool
        \@@_create_medium_and_large_nodes:
        \@@_create_medium_nodes:
    }
    { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
  \int_zero_new:N \l_@@_initial_i_int
  \int_zero_new:N \l_@@_initial_j_int
  \int_zero_new:N \l_@@_final_i_int
  \int_zero_new:N \l_@@_final_j_int
  \bool_set_false:N \l_@@_initial_open_bool
  \bool_set_false:N \l_@@_final_open_bool
```

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

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

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

```
      \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
    }
```

Now, we actually draw the dotted lines.

```
  \@@_draw_dotted_lines:
  \bool_if:NTF \l_@@_hvlines_bool
    \@@_draw_hvlines:
    {
      \bool_if:NT \l_@@_hlines_bool \@@_draw_hlines:
      \bool_if:NT \l_@@_vlines_bool \@@_draw_vlines:
    }
```

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

```
  \cs_set_eq:NN \ialign \@@_old_ialign:
  \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
  \g_@@_internal_code_after_tl
  \tl_gclear:N \g_@@_internal_code_after_tl
  \bool_if:NT \c_@@_tikz_loaded_bool
    {
      \tikzset
        {
          every~picture / .style =
            {
              overlay ,
              remember~picture ,
              name~prefix = \@@_env: -
            }
        }
    }
  \cs_set_eq:NN \line \@@_line
```

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 one is eventually present in `\g_@@_code_after_tl`. That's why we set `\Code-after` to be *no-op* now.

```
    \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

And here's the `code-after`:

```
    \g_@@_code_after_tl
    \tl_gclear:N \g_@@_code_after_tl
    \group_end:
    \str_gclear:N \g_@@_name_env_str
    \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[38]. This command is used by `\CT@arc@` but we use it also for compatibility with colortbl. But we want also to be able to use color for the rules of the array when colortbl is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by colortbl.

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

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly "visible". That's why we have to define the adequate version of `\@@_draw_dotted_lines:` whether Tikz is loaded or not (in that case, only PGF is loaded).

```
\AtBeginDocument
  {
    \cs_new_protected:Npx \@@_draw_dotted_lines:
      {
        \c_@@_pgfortikzpicture_tl
        \@@_draw_dotted_lines_i:
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

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

\cs_new_protected:Npn \@@_restore_iRow_jCol:
  {
    \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
    \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
  }
```

We draw the dotted lines

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

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

[38] e.g. `\color[rgb]{0.5,0.5,0}`)

- 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.

```
\cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
 {
```

First, we declare the current cell as "dotted" because we forbide intersections of dotted lines.

```
    \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

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

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

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

We test if we are still in the matrix.

```
        \bool_set_false:N \l_@@_final_open_bool
        \int_compare:nNnTF \l_@@_final_i_int > \c@iRow
          {
            \int_compare:nNnTF { #3 } = 1
              { \bool_set_true:N \l_@@_final_open_bool }
              {
                \int_compare:nNnT \l_@@_final_j_int > \c@jCol
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
          }
          {
            \int_compare:nNnTF \l_@@_final_j_int < 1
              {
                \int_compare:nNnT { #4 } = { -1 }
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
              {
                \int_compare:nNnT \l_@@_final_j_int > \c@jCol
                  {
                    \int_compare:nNnT { #4 } = 1
                      { \bool_set_true:N \l_@@_final_open_bool }
                  }
              }
          }
        \bool_if:NTF \l_@@_final_open_bool
```

If we are outside the matrix, we have found the extremity of the dotted line and it's an *open* extremity.

```
        {
```

We do a step backwards.

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

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

```
        {
          \cs_if_exist:cTF
            {
              @@ _ dotted _
              \int_use:N \l_@@_final_i_int -
              \int_use:N \l_@@_final_j_int
            }
            {
              \int_sub:Nn \l_@@_final_i_int { #3 }
              \int_sub:Nn \l_@@_final_j_int { #4 }
              \bool_set_true:N \l_@@_final_open_bool
              \bool_set_true:N \l_@@_stop_loop_bool
            }
            {
              \cs_if_exist:cTF
                {
                  pgf @ sh @ ns @ \@@_env:
                  - \int_use:N \l_@@_final_i_int
                  - \int_use:N \l_@@_final_j_int
                }
                { \bool_set_true:N \l_@@_stop_loop_bool }
```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be mark as "dotted" because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environnement), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```
              {
                \cs_set:cpn
                  {
                    @@ _ dotted _
                    \int_use:N \l_@@_final_i_int -
                    \int_use:N \l_@@_final_j_int
                  }
                  { }
              }
            }
        }
    }
```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programmation is similar to the previous one.

```
    \bool_set_false:N \l_@@_stop_loop_bool
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
        \int_sub:Nn \l_@@_initial_i_int { #3 }
        \int_sub:Nn \l_@@_initial_j_int { #4 }
        \bool_set_false:N \l_@@_initial_open_bool
        \int_compare:nNnTF \l_@@_initial_i_int < 1
          {
            \int_compare:nNnTF { #3 } = 1
```

```
            { \bool_set_true:N \l_@@_initial_open_bool }
            {
              \int_compare:nNnT \l_@@_initial_j_int = 0
                { \bool_set_true:N \l_@@_initial_open_bool }
            }
        }
        {
          \int_compare:nNnTF \l_@@_initial_j_int < 1
            {
              \int_compare:nNnT { #4 } = 1
                { \bool_set_true:N \l_@@_initial_open_bool }
            }
            {
              \int_compare:nNnT \l_@@_initial_j_int > \c@jCol
                {
                  \int_compare:nNnT { #4 } = { -1 }
                    { \bool_set_true:N \l_@@_initial_open_bool }
                }
            }
        }
      \bool_if:NTF \l_@@_initial_open_bool
        {
          \int_add:Nn \l_@@_initial_i_int { #3 }
          \int_add:Nn \l_@@_initial_j_int { #4 }
          \bool_set_true:N \l_@@_stop_loop_bool
        }
        {
          \cs_if_exist:cTF
            {
              @@ _ dotted _
              \int_use:N \l_@@_initial_i_int -
              \int_use:N \l_@@_initial_j_int
            }
            {
              \int_add:Nn \l_@@_initial_i_int { #3 }
              \int_add:Nn \l_@@_initial_j_int { #4 }
              \bool_set_true:N \l_@@_initial_open_bool
              \bool_set_true:N \l_@@_stop_loop_bool
            }
            {
              \cs_if_exist:cTF
                {
                  pgf @ sh @ ns @ \@@_env:
                  - \int_use:N \l_@@_initial_i_int
                  - \int_use:N \l_@@_initial_j_int
                }
                { \bool_set_true:N \l_@@_stop_loop_bool }
                {
                  \cs_set:cpn
                    {
                      @@ _ dotted _
                      \int_use:N \l_@@_initial_i_int -
                      \int_use:N \l_@@_initial_j_int
                    }
                    { }
                }
            }
        }
    }
```

If the key `hvlines` is used, 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.

```
    \bool_if:NT \l_@@_hvlines_bool
      {
```

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

\cs_new_protected:Npn \@@_set_initial_coords:
  {
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_final_coords:
  {
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_initial_i_int
        - \int_use:N \l_@@_initial_j_int
      }
      { #1 }
    \@@_set_initial_coords:
  }
\cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_final_i_int
        - \int_use:N \l_@@_final_j_int
      }
      { #1 }
    \@@_set_final_coords:
  }
```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

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

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

```
        \group_begin:
          \int_compare:nNnTF { #1 } = 0
            { \color { nicematrix-first-row } }
            {
```

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

```
              \int_compare:nNnT { #1 } = \l_@@_last_row_int
                { \color { nicematrix-last-row } }
```

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

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

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

The following function is also used by `\Hdotsfor`.

```
\cs_new_protected:Npn \@@_actually_draw_Ldots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_add:Nn \l_@@_x_initial_dim
           { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
      }
      { \@@_set_initial_coords_from_anchor:n { base-east } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
        \dim_sub:Nn \l_@@_x_final_dim
           { \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep }
        \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
      }
      { \@@_set_final_coords_from_anchor:n { base-west } }
```

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

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

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

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

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

```
        \group_begin:
          \int_compare:nNnTF { #1 } = 0
            { \color { nicematrix-first-row } }
            {
```

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

```
              \int_compare:nNnT { #1 } = \l_@@_last_row_int
                { \color { nicematrix-last-row } }
            }
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Cdots:
        \group_end:
      }
  }
```

The command `\@@_actually_draw_Cdots:` has the following implicit arguments:

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

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

```
          { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
      }
    \@@_draw_line:
  }
```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```
\cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
  {
    \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 0
```

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

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

The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

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

The following function is also used by `\Vdotsfor`.

```
\cs_new_protected:Npn \@@_actually_draw_Vdots:
  {
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` (`L` of `{NiceArray}`) or may be considered as if.

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

Now, we try to determine whether the column is of type `c` (`C` of `{NiceArray}`) or may be considered as if.

```
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
      }
```

```
      { \@@_set_initial_coords_from_anchor:n { south } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
      }
      { \@@_set_final_coords_from_anchor:n { north } }
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
            \dim_set_eq:NN \l_tmpa_dim \pgf@x
            \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
            \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
            \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
```

We may think that the final user won't use a "last column" which contains only a command `\Vdots`. However, if the `\Vdots` is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

```
            \int_compare:nNnT \l_@@_last_col_int > { -2 }
              {
                \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
                  {
                    \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
                    \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
                    \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
                    \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
                  }
              }
          }
          { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
      }
      {
        \bool_if:NTF \l_@@_final_open_bool
          { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
          {
```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` (`C` of `{NiceArray}`) or may be considered as if.

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

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.
The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```
\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
  {
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
```

```
      {
        \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 1
```

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

```
        \group_begin:
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Ddots:
        \group_end:
      }
  }
```

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

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

```
\cs_new_protected:Npn \@@_actually_draw_Ddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
        \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      }
      { \@@_set_initial_coords_from_anchor:n { south~east } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
        \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
      }
      { \@@_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.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gincr:N \g_@@_ddots_int
```

We test if the diagonal line is the first one (the counter `\g_@@_ddots_int` is created for this usage).

```
        \int_compare:nNnTF \g_@@_ddots_int = 1
```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the 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.

```
          {
            \dim_gset:Nn \g_@@_delta_x_one_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_one_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
```

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate \l_@@_x_initial_dim.

```
          {
            \dim_set:Nn \l_@@_y_final_dim
              {
                \l_@@_y_initial_dim +
                ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
              }
          }
      }
    \@@_draw_line:
  }
```

We draw the \Iddots diagonals in the same way.
The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
  {
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
```

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

```
        \group_begin:
          \keys_set:nn { NiceMatrix / xdots } { #3 }
          \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \@@_actually_draw_Iddots:
        \group_end:
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_actually_draw_Iddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
        \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      }
      { \@@_set_initial_coords_from_anchor:n { south~west } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_qpoint:n { row - \@@_succ:n \l_@@_final_i_int }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
        \@@_qpoint:n { col - \int_use:N \l_@@_final_j_int }
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
      }
```

```
      { \@@_set_final_coords_from_anchor:n { north~east } }
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gincr:N \g_@@_iddots_int
        \int_compare:nNnTF \g_@@_iddots_int = 1
          {
            \dim_gset:Nn \g_@@_delta_x_two_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_two_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
          {
            \dim_set:Nn \l_@@_y_final_dim
              {
                \l_@@_y_initial_dim +
                ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
              }
          }
      }
    \@@_draw_line:
  }
```

The actual instructions for drawing the dotted line with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```
\cs_new_protected:Npn \@@_draw_line:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_standard_tl
      \@@_draw_standard_dotted_line:
      \@@_draw_non_standard_dotted_line:
  }
```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```
\cs_new_protected:Npn \@@_draw_non_standard_dotted_line:
  {
     \begin { scope }
     \exp_args:No \@@_draw_non_standard_dotted_line:n
       { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
  }
```

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 diredtly `\l_@@_xdots_color_tl`).

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

```
\cs_new_protected:Npn \@@_draw_non_standard_dotted_line:n #1
  {
```

```
    \draw
      [
        #1 ,
        shorten~> = \l_@@_xdots_shorten_dim ,
        shorten~< = \l_@@_xdots_shorten_dim ,
      ]
          ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
       -- node [ sloped , above ]
            { \c_math_toggle_token \scriptstyle \l_@@_xdots_up_tl \c_math_toggle_token }
          node [ sloped , below ]
            {
              \c_math_toggle_token
              \scriptstyle \l_@@_xdots_down_tl
              \c_math_toggle_token
            }
          ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
    \end { scope }
  }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of points (which give a dotted line with real round points).

```
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
  {
```

First, we put the labels.

```
    \bool_lazy_and:nnF
      { \tl_if_empty_p:N \l_@@_xdots_up_tl }
      { \tl_if_empty_p:N \l_@@_xdots_down_tl }
      {
        \pgfscope
        \pgftransformshift
          {
            \pgfpointlineattime { 0.5 }
              { \pgfpoint \l_@@_x_initial_dim  \l_@@_y_initial_dim }
              { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
          }
        \pgftransformrotate
          {
            \fp_eval:n
              {
                atand
                 (
                  \l_@@_y_final_dim - \l_@@_y_initial_dim ,
                   \l_@@_x_final_dim - \l_@@_x_initial_dim
                  )
              }
          }
        \pgfnode
          { rectangle }
          { south }
          {
            \c_math_toggle_token
            \scriptstyle \l_@@_xdots_up_tl
            \c_math_toggle_token
          }
          { }
          { \pgfusepath { } }
        \pgfnode
          { rectangle }
          { north }
          {
            \c_math_toggle_token
            \scriptstyle \l_@@_xdots_down_tl
```

```
            \c_math_toggle_token
          }
          { }
          { \pgfusepath { } }
        \endpgfscope
      }
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \group_begin:
```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of expl3 to compute this length.

```
      \dim_zero_new:N \l_@@_l_dim
      \dim_set:Nn \l_@@_l_dim
        {
          \fp_to_dim:n
            {
              sqrt
               (
                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
                    +
                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
               )
            }
        }
```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

```
      \bool_lazy_or:nnF
        { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
        { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
        \@@_draw_standard_dotted_line_i:
    \group_end:
  }
\dim_const:Nn \c_@@_max_l_dim { 50 cm }
\cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
  {
```

The integer `\l_tmpa_int` is the number of dots of the dotted line.

```
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \int_set:Nn \l_tmpa_int
              { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
          }
          {
            \int_set:Nn \l_tmpa_int
              {
                \dim_ratio:nn
                  { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
                  \l_@@_inter_dots_dim
              }
          }
      }
      {
        \bool_if:NTF \l_@@_final_open_bool
          {
            \int_set:Nn \l_tmpa_int
              {
                \dim_ratio:nn
```

```
                { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
                \l_@@_inter_dots_dim
            }
        }
        {
          \int_set:Nn \l_tmpa_int
            {
              \dim_ratio:nn
                { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim  }
                \l_@@_inter_dots_dim
            }
        }
    }
```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```
    \dim_set:Nn \l_tmpa_dim
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_tmpb_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
      }
```

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`.

```
    \int_set:Nn \l_tmpb_int
      {
        \bool_if:NTF \l_@@_initial_open_bool
          { \bool_if:NTF \l_@@_final_open_bool 1 0 }
          { \bool_if:NTF \l_@@_final_open_bool 2 1 }
      }
```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```
    \dim_gadd:Nn \l_@@_x_initial_dim
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn
          { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
          { 2 \l_@@_l_dim }
        * \l_tmpb_int
      }
    \dim_gadd:Nn \l_@@_y_initial_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn
          { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
          { 2 \l_@@_l_dim }
        * \l_tmpb_int
      }
    \pgf@relevantforpicturesizefalse
    \int_step_inline:nnn 0 \l_tmpa_int
      {
        \pgfpathcircle
          { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
          { \l_@@_radius_dim }
        \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
        \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
```

```
      }
    \pgfusepathqfill
  }
```

User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of nicematrix rely upon `{NiceArray}`).
The starred versions of these commands are deprecated since version 3.1 but, as for now, they are still available with an error.

The syntax of these commands uses the character `_` as embellishment and thats' why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.

```
\AtBeginDocument
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } E { _ ^ } { { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
    \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@jCol = 0
          { \@@_error:nn { in~first~col } \Ldots }
          {
            \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
              { \@@_error:nn { in~last~col } \Ldots }
              {
                 \@@_instruction_of_type:nn { Ldots }
                   { #1 , down = #2 , up = #3 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ldots }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@jCol = 0
          { \@@_error:nn { in~first~col } \Cdots }
          {
            \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
              { \@@_error:nn { in~last~col } \Cdots }
              {
                 \@@_instruction_of_type:nn { Cdots }
                   { #1 , down = #2 , up = #3 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_cdots }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
      {
        \int_compare:nNnTF \c@iRow = 0
          { \@@_error:nn { in~first~row } \Vdots }
```

```
        {
          \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
            { \@@_error:nn { in~last~row } \Vdots }
            {
               \@@_instruction_of_type:nn { Vdots }
                 { #1 , down = #2 , up = #3 }
            }
        }
      \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_vdots }
      \bool_gset_true:N \g_@@_empty_cell_bool
    }


  \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
    {
      \int_case:nnF \c@iRow
        {
          0                  { \@@_error:nn { in~first~row } \Ddots }
          \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
        }
        {
          \int_case:nnF \c@jCol
            {
              0                  { \@@_error:nn { in~first~col } \Ddots }
              \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
            }
            {
              \@@_instruction_of_type:nn { Ddots }
                { #1 , down = #2 , up = #3 }
            }

        }
      \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_ddots }
      \bool_gset_true:N \g_@@_empty_cell_bool
    }


  \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
    {
      \int_case:nnF \c@iRow
        {
          0                  { \@@_error:nn { in~first~row } \Iddots }
          \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
        }
        {
          \int_case:nnF \c@jCol
            {
              0                  { \@@_error:nn { in~first~col } \Iddots }
              \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
            }
            {
              \@@_instruction_of_type:nn { Iddots }
                { #1 , down = #2 , up = #3 }
            }

        }
      \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_old_iddots }
      \bool_gset_true:N \g_@@_empty_cell_bool
    }
}
```

End of the `\AtBeginDocument`.

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

```
\cs_new_protected:Npn \@@_Hspace:
  {
   \bool_gset_true:N \g_@@_empty_cell_bool
   \hspace
  }
```

In the environment {NiceArray}, the command \multicolumn will be linked to the following command \@@_multicolumn:nnn.

```
\cs_set_eq:NN \@@_old_multicolumn \multicolumn
\cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
  {
    \@@_old_multicolumn { #1 } { #2 } { #3 }
```

The \peek_remove_spaces:n is mandatory.

```
    \peek_remove_spaces:n
      {
        \int_compare:nNnT #1 > 1
          {
            \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
            \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
            \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
              {
                { \int_use:N \c@iRow }
                { \int_use:N \c@jCol }
                { \int_use:N \c@iRow }
                { \int_eval:n { \c@jCol + #1 - 1 } }
              }
          }
        \int_gadd:Nn \c@jCol { #1 - 1 }
     }
  }
```

The command \@@_Hdotsfor will be linked to \Hdotsfor in {NiceArrayWithDelims}. Tikz nodes are created also in the implicit cells of the \Hdotsfor (maybe we should modify that point).

This command must *not* be protected since it begins with \multicolumn.

```
\cs_new:Npn \@@_Hdotsfor:
  {
    \multicolumn { 1 } { C } { }
    \@@_Hdotsfor_i
  }
```

The command \@@_Hdotsfor_i is defined with the tools of xparse because it has an optional argument. Note that such a command defined by \NewDocumentCommand is protected and that's why we have put the \multicolumn before (in the definition of \@@_Hdotsfor:).

```
\AtBeginDocument
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
```

We don't put ! before the last optionnal argument for homogeneity with \Cdots, etc. which have only one optional argument.

```
    \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
      {
        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
          {
            \@@_Hdotsfor:nnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { #2 }
              {
                #1 , #3 ,
```

```
                down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
              }
          }
        \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } }
      }
  }
```

Enf of `\AtBeginDocument`.

```
\cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
    \int_set:Nn \l_@@_initial_i_int { #1 }
    \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
    \int_compare:nNnTF #2 = 1
      {
        \int_set:Nn \l_@@_initial_j_int 1
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_use:N \l_@@_initial_i_int
            - \int_eval:n { #2 - 1 }
          }
          { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
          {
            \int_set:Nn \l_@@_initial_j_int { #2 }
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
      {
        \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_use:N \l_@@_final_i_int
            - \int_eval:n { #2 + #3 }
          }
          { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
          {
            \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
            \bool_set_true:N \l_@@_final_open_bool
          }
      }
    \group_begin:
    \int_compare:nNnTF { #1 } = 0
      { \color { nicematrix-first-row } }
      {
        \int_compare:nNnT { #1 } = \g_@@_row_total_int
          { \color { nicematrix-last-row } }
      }
    \keys_set:nn { NiceMatrix / xdots } { #4 }
    \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
```

```
    \@@_actually_draw_Ldots:
    \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).

```
    \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
      { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
  }


\AtBeginDocument
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
    \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
      {
        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
          {
            \@@_Vdotsfor:nnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { #2 }
              {
                #1 , #3 ,
                down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
              }
          }
      }
  }
```

Enf of `\AtBeginDocument`.

```
\cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

For the column, it's easy.

```
    \int_set:Nn \l_@@_initial_j_int { #2 }
    \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```
    \int_compare:nNnTF #1 = 1
      {
        \int_set:Nn \l_@@_initial_i_int 1
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 - 1 }
            - \int_use:N \l_@@_initial_j_int
          }
          { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
          {
            \int_set:Nn \l_@@_initial_i_int { #1 }
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #1 + #3 -1 } = \c@iRow
      {
        \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
```

```
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 + #3 }
            - \int_use:N \l_@@_final_j_int
          }
          { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
          {
            \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
            \bool_set_true:N \l_@@_final_open_bool
          }
      }
    \group_begin:
    \int_compare:nNnTF { #2 } = 0
      { \color { nicematrix-first-col } }
      {
        \int_compare:nNnT { #2 } = \g_@@_col_total_int
          { \color { nicematrix-last-col } }
      }
    \keys_set:nn { NiceMatrix / xdots } { #4 }
    \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
    \@@_actually_draw_Vdots:
    \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).

```
    \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
      { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
  }
```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.
The command will exit three levels of groups (only two in `{NiceTabular}` because there is not the group of the math mode to exit) in order to execute the command
"`\box_rotate:Nn \l_@@_cell_box { 90 }`"
just after the construction of the box `\l_@@_cell_box`.

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

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

```
    \int_compare:nNnT \c@iRow = \l_@@_last_row_int
      {
        \vbox_set_top:Nn \l_@@_cell_box
          {
            \vbox_to_zero:n { }
```

`0.8 ex` will be the distance between the principal part of the array and our element (which is composed with `\rotate`.

```
            \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
            \box_use:N \l_@@_cell_box
```

```
        }
    }
}
```

The command \line accessible in code-after

In the `code-after`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i*-*j*) and draws a dotted line between these cells.

First, we write a command with an argument of the format *i*-*j* and applies the command `\int_eval:n` to *i* and *j* ; this must *not* be protected (and is, of course fully expandable).[39]

```
\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
  { \int_eval:n { #1 } - \int_eval:n { #2 } }
```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```
\AtBeginDocument
  {
    \tl_set:Nn \l_@@_argspec_tl { O { } m m ! O { } E { _ ^ } { { } { } } }
    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
    \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
      {
        \group_begin:
        \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
        \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
          \use:x
            {
              \@@_line_i:nn
                { \@@_double_int_eval:n #2 \q_stop }
                { \@@_double_int_eval:n #3 \q_stop }
            }
        \group_end:
      }
  }
\cs_new_protected:Npn \@@_line_i:nn #1 #2
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
    \bool_if:nTF
      {
        \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
          ||
        \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
      }
      {
        \@@_error:nnn { unknown~cell~for~line~in~code-after } { #1 } { #2 }
      }
      { \@@_draw_line_ii:nn { #1 } { #2 } }
  }
\AtBeginDocument
  {
    \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
      {
```

[39] Indeed, we want that the user may use the command `\line` in `code-after` with LaTeX counters in the arguments — with the command `\value`.

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly "visible" and that why we do this static construction of the command `\@@_draw_line_ii:`.

```
        \c_@@_pgfortikzpicture_tl
        \@@_draw_line_iii:nn { #1 } { #2 }
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```
\cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
  {
    \pgfrememberpicturepositiononpagetrue
    \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \@@_draw_line:
  }
```

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).

Commands available in the code-before

In the beginning of the `code-before`, the command `\@@_rowcolor:nn` will be linked to `\rowcolor` and the command `\@@_columncolor:nn` to `\columncolor`.

```
\cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_set:Nn \l_tmpb_tl { #2 }
  }
```

Here an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

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

`\l_tmpa_dim` is the x-value of the right side of the rows.

```
        \@@_qpoint:n { col - 1}
        \int_compare:nNnTF \l_@@_first_col_int = 0
          { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
          { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
        \@@_qpoint:n { col - \@@_succ:n \c@jCol }
        \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
        \clist_map_inline:nn { #3 }
          {
            \tl_set:Nn \l_tmpa_tl { ##1 }
            \tl_if_in:NnTF \l_tmpa_tl { - }
              { \@@_cut_on_hyphen:w ##1 \q_stop }
              { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
            \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
            \tl_if_empty:NT \l_tmpb_tl
              { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
```

```
            \int_compare:nNnT \l_tmpb_tl > \c@iRow
              { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
```

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

```
            \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
            \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
            \@@_qpoint:n { row - \l_tmpa_tl }
            \dim_set:Nn \l_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
            \pgfpathrectanglecorners
              { \pgfpoint \l_tmpc_dim \l_tmpd_dim }
              { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
          }
        \pgfusepathqfill
        \endpgfpicture
      }
  }
```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

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

`\l_tmpa_dim` is the y-value of the top of the columns et `\l_tmpb_dim` is the y-value of the bottom.

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

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

```
            \@@_qpoint:n { col - \l_tmpa_tl }
            \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
              { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
              { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
            \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
            \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
            \pgfpathrectanglecorners
              { \pgfpoint \l_tmpc_dim \l_tmpa_dim }
              { \pgfpoint \l_tmpd_dim \l_tmpb_dim }
          }
        \pgfusepathqfill
        \endpgfpicture
      }
  }
```

Here an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
\NewDocumentCommand \@@_cellcolor { O { } m m }
  {
    \tl_if_blank:nF { #2 }
```

```
    {
      \pgfpicture
      \pgf@relevantforpicturesizefalse
      \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
      \clist_map_inline:nn { #3 }
        {
          \@@_cut_on_hyphen:w ##1 \q_stop
          \@@_qpoint:n { row - \l_tmpa_tl }
          \bool_lazy_and:nnT
            { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
            { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
            {
              \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
              \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
              \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
              \@@_qpoint:n { col - \l_tmpb_tl }
              \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
                { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
                { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
              \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
              \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
              \pgfpathrectanglecorners
                { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
                { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
            }
        }
      \pgfusepathqfill
      \endpgfpicture
    }
}
```

Here an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
\NewDocumentCommand \@@_rectanglecolor { O { } m m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \pgfpicture
        \pgf@relevantforpicturesizefalse
        \tl_if_empty:nTF { #1 } \color { \color [ #1 ] } { #2 }
        \@@_cut_on_hyphen:w #3 \q_stop
        \bool_lazy_and:nnT
          { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
          { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
          {
            \@@_qpoint:n { row - \l_tmpa_tl }
            \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
            \@@_qpoint:n { col - \l_tmpb_tl }
            \int_compare:nNnTF \l_@@_first_col_int = \l_tmpb_tl
              { \dim_set:Nn \l_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
              { \dim_set:Nn \l_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
            \@@_cut_on_hyphen:w #4 \q_stop
            \int_compare:nNnT \l_tmpa_tl > \c@iRow
              { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
            \int_compare:nNnT \l_tmpb_tl > \c@jCol
              { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
            \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
            \dim_set:Nn \l_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
            \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
            \dim_set:Nn \l_tmpd_dim { \pgf@x + 0.5 \arrayrulewidth }
            \pgfpathrectanglecorners
              { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
              { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
            \pgfusepathqfill
```

```
          }
        \endpgfpicture
      }
  }
```

The command `\rowcolors` (accessible in the `code-before`) is inspired by the command `\rowcolors` of the package xcolor (with the option `table`). However, the command `\rowcolors` of nicematrix has *not* the optional argument of the command `\rowcolors` of xcolor.

```
\NewDocumentCommand \@@_rowcolors { O { } m m m }
  {
    \int_step_inline:nnn { #2 } { \int_use:N \c@iRow }
      {
        \int_if_odd:nTF { ##1 }
          { \@@_rowcolor [ #1 ] { #3 } }
          { \@@_rowcolor [ #1 ] { #4 } }
        { ##1 }
      }
  }


\NewDocumentCommand \@@_chessboardcolors { O { } m m  }
  {
    \int_step_inline:nn { \int_use:N \c@iRow }
      {
        \int_step_inline:nn { \int_use:N \c@jCol }
          {
            \int_if_even:nTF { ####1 + ##1 }
              { \@@_cellcolor [ #1 ] { #2 } }
              { \@@_cellcolor [ #1 ] { #3 } }
            { ##1 - ####1 }
          }
      }
  }
```

The vertical rules

We give to the user the possibility to define new types of columns (with `\newcolumntype` of array) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.
We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of nicematrix (and so the user will be allowed to use the same new type of column in the environments of nicematrix and in the standard environments of array).
That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
\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`.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
  {
    \int_compare:nNnTF \l_@@_first_col_int = 0
      { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      {
        \int_compare:nNnTF \c@jCol = 0
          {
            \int_compare:nNnF \c@iRow = { -1 }
              { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
          }
          { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      }
  }
```

This definition may seem complicated by we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* an potential vertical rule on the left side of the first cell.

The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
  {
    \int_compare:nNnF \c@iRow = 0
      { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
  }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

In fact, independently of `\OnlyMainNiceMatrix`, which is a convenience given to the user, we have to modify the behaviour of the standard specifier "`|`".
Remark first that the natural way to do that would be to redefine the specifier "`|`" with `\newcolumntype`:

```
\newcolumntype { | } { ! { \OnlyMainNiceMatrix \vline } }
```

However, this code fails if the user uses `\DefineShortVerb{\|}` of fancyvrb. Moreover, it would not be able to deal correctly with two consecutive specifiers "`|`" (in a preambule like `ccc||ccc`).
That's why we have done a redefinition of the macro `\@arrayrule` of array and this redefinition will add `\@@_vline:` instead of `\vline` in the preamble (that definition is in the beginning of `{NiceArrayWithDelims}`).
Here is the definition of `\@@_vline:`. This definition *must* be protected because you don't want that macro expanded during the construction of the preamble (the tests in `\@@_OnlyMainNiceMatrix:n` must be effective in each row and not once for all when the preamble is constructed). 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.

```
\cs_new_protected:Npn \@@_vline:
  { \@@_OnlyMainNiceMatrix:n { { \CT@arc@ \vline } } }
```

The command `\@@_draw_vlines` will be executed when the user uses the option `vlines` (which draws all the vlines of the array).

```
\cs_new_protected:Npn \@@_draw_vlines:
  {
    \group_begin:
```

The command `\CT@arc@` is a command of colortbl which sets the color of the rules in the array. The package nicematrix uses it even if colortbl is not loaded.

```
    \CT@arc@
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \pgfsetlinewidth \arrayrulewidth
```

First, we compute in `\l_tmpa_dim` the height of the rules we have to draw.

```
    \@@_qpoint:n { row - 1 }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { row - \@@_succ:n \c@iRow }
    \dim_sub:Nn \l_tmpa_dim \pgf@y
```

We translate vertically to take into account the potential "last row".

```
    \dim_zero:N \l_tmpb_dim
    \int_compare:nNnT \l_@@_last_row_int > { -1 }
      {
        \dim_set_eq:NN \l_tmpb_dim \g_@@_dp_last_row_dim
        \dim_add:Nn \l_tmpb_dim \g_@@_ht_last_row_dim
```

We adjust the value of \l_tmpa_dim by the width of the horizontal rule just before the “last row”.

```
        \@@_qpoint:n { row - \@@_succ:n \c@iRow }
        \dim_add:Nn \l_tmpa_dim \pgf@y
        \@@_qpoint:n { row - \@@_succ:n \g_@@_row_total_int }
        \dim_sub:Nn \l_tmpa_dim \pgf@y
        \dim_sub:Nn \l_tmpa_dim \l_tmpb_dim
      }
    \dim_add:Nn \l_tmpa_dim \arrayrulewidth
```

Now, we can draw the vertical rules with a loop.

```
    \int_step_inline:nnn
      { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
      { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@jCol } \c@jCol }
      {
        \pgfpathmoveto { \@@_qpoint:n { col - ##1 } }
        \pgfpathlineto
          {
            \pgfpointadd
              { \@@_qpoint:n { col - ##1 } }
              { \pgfpoint \c_zero_dim { \l_tmpb_dim + \l_tmpa_dim } }
          }
      }
    \pgfusepathqstroke
    \endpgfpicture
    \group_end:
  }
```

The key hvlines

```
\cs_new_protected:Npn \@@_draw_hlines:
  {
    \pgfpicture
    \CT@arc@
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \pgfsetlinewidth \arrayrulewidth
    \int_step_inline:nnn
      { \bool_if:NTF \l_@@_NiceArray_bool 1 2 }
      { \bool_if:NTF \l_@@_NiceArray_bool { \@@_succ:n \c@iRow } \c@iRow }
      {
        \@@_qpoint:n { row - ##1 }
        \dim_set_eq:NN \l_tmpa_dim \pgf@y
        \pgfpathmoveto { \pgfpoint \pgf@x \pgf@y }
        \@@_qpoint:n { col - \@@_succ:n { \c@jCol } }
        \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \arrayrulewidth }
        \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
      }
    \pgfusepathqstroke
    \endpgfpicture
  }
```

Since version 4.1, the key `hvlines` is no longer a mere alias for the conjonction of `hlines` and `vlines`. Indeed, with `hvlines`, the vertical and horizontal rules are *not* drawn within the blocks (created by `\Block`) nor within the “virtual blocks” (corresponding to the dotted lines drawn by `\Cdots`, `\Vdots`, etc.).

```
\cs_new_protected:Npn \@@_draw_hvlines:
  {
    \bool_lazy_and:nnTF
     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
     \@@_draw_hvlines_i:
     \@@_draw_hvlines_ii:
```

```
  }
```

This version is only for efficiency. The general case (in `\@@_draw_hvlines_ii:`) does the job in all case (but slower).

```
\cs_new_protected:Npn \@@_draw_hvlines_i:
  {
    \@@_draw_hlines:
    \@@_draw_vlines:
  }
```

Now, the general case, where there are blocks or dots in the array.

```
\cs_new_protected:Npn \@@_draw_hvlines_ii:
  {
    \group_begin:
    \CT@arc@
```

First, the exterior rectangle of the array (only in `{NiceArray}` and `{NiceTabular}`).

```
    \bool_if:NT \l_@@_NiceArray_bool
      {
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgf@relevantforpicturesizefalse
        \pgfsetlinewidth \arrayrulewidth
        \pgfsetrectcap
        \@@_qpoint:n { col - 1 }
        \dim_set_eq:NN \l_tmpa_dim \pgf@x
        \@@_qpoint:n { row -1 }
        \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \@@_qpoint:n { col - \@@_succ:n \c@jCol }
        \dim_set_eq:NN \l_tmpc_dim \pgf@x
        \@@_qpoint:n { row - \@@_succ:n \c@iRow }
        \pgfpathrectanglecorners
          { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
          { \pgfpoint \l_tmpc_dim \pgf@y }
        \pgfusepathqstroke
        \endpgfpicture
      }
```

First, the horizontal rules in the interior of the array.

```
    \int_step_variable:nnNn 2 \c@iRow \l_tmpa_tl
      {
        \int_step_variable:nNn \c@jCol \l_tmpb_tl
          {
```

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.

```
            \bool_gset_true:N \g_tmpa_bool
            \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
              { \@@_test_if_hline_in_block:nnnn ##1 }
            \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
              { \@@_test_if_hline_in_block:nnnn ##1 }
            \bool_if:NT \g_tmpa_bool
              {
                \pgfpicture
                \pgfrememberpicturepositiononpagetrue
                \pgf@relevantforpicturesizefalse
                \pgfsetlinewidth \arrayrulewidth
                \pgfsetrectcap
                \@@_qpoint:n { row - \l_tmpa_tl }
                \dim_set_eq:NN \l_tmpb_dim \pgf@y
                \@@_qpoint:n { col - \l_tmpb_tl }
                \dim_set_eq:NN \l_tmpa_dim \pgf@x
```

```
                \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
                \dim_set_eq:NN \l_tmpc_dim \pgf@x
                \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
                \pgfpathlineto { \pgfpoint \l_tmpc_dim \l_tmpb_dim }
                \pgfusepathqstroke
                \endpgfpicture
              }
          }
      }
```

Now, the vertical rules in the interior of the array.

```
    \int_step_variable:nNn \c@iRow \l_tmpa_tl
      {
        \int_step_variable:nnNn 2 \c@jCol \l_tmpb_tl
          {
```

The boolean `\g_tmpa_bool` indicates whether the small vertial 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 vertial rule won't be drawn.

```
            \bool_gset_true:N \g_tmpa_bool
            \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
              { \@@_test_if_vline_in_block:nnnn ##1 }
            \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
              { \@@_test_if_vline_in_block:nnnn ##1 }
            \bool_if:NT \g_tmpa_bool
              {
                \pgfpicture
                \pgfrememberpicturepositiononpagetrue
                \pgf@relevantforpicturesizefalse
                \pgfsetlinewidth \arrayrulewidth
                \pgfsetrectcap
                \@@_qpoint:n { row - \l_tmpa_tl }
                \dim_set_eq:NN \l_tmpb_dim \pgf@y
                \@@_qpoint:n { col - \l_tmpb_tl }
                \dim_set_eq:NN \l_tmpa_dim \pgf@x
                \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
                \dim_set_eq:NN \l_tmpc_dim \pgf@y
                \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
                \pgfpathlineto { \pgfpoint \l_tmpa_dim \l_tmpc_dim }
                \pgfusepathqstroke
                \endpgfpicture
              }
          }
      }
```

The group was for the color of the rules.

```
    \group_end:
    \seq_gclear:N \g_@@_pos_of_xdots_seq
  }
```

The following command tests wether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the col) would provide an horizontal rule towards the right in the block delimited by the four arguments `#1`, `#2`, `#3` and `#4`. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to `false`.

```
\cs_set_protected:Npn \@@_test_if_hline_in_block:nnnn #1 #2 #3 #4
  {
    \int_compare:nNnT \l_tmpa_tl > { #1 }
      {
        \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
          {
            \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
              {
                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
```

```
        }
    }
}
```

The same for vertical rules.

```
\cs_set_protected:Npn \@@_test_if_vline_in_block:nnnn #1 #2 #3 #4
  {
    \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
      {
        \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
          {
            \int_compare:nNnT \l_tmpb_tl > { #2 }
              {
                \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
          }
      }
  }
```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```
\cs_new:Npn \@@_hdottedline:
  {
    \noalign { \skip_vertical:N 2\l_@@_radius_dim }
    \@@_hdottedline_i:
  }
```

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

```
\cs_new_protected:Npn \@@_hdottedline_i:
  {
```

We write in the code-after the instruction that will eventually 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).

```
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      { \@@_hdottedline:n { \int_use:N \c@iRow } }
  }
```

The command `\@@_hdottedline:n` is the command written in the `code-after` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```
\AtBeginDocument
  {
```

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

```
    \cs_new_protected:Npx \@@_hdottedline:n #1
      {
        \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
        \bool_set_true:N \exp_not:N \l_@@_final_open_bool
        \c_@@_pgfortikzpicture_tl
        \@@_hdottedline_i:n { #1 }
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

The following command *must* be protected since it is used in the construction of `\@@_hdottedline:n`.

```
\cs_new_protected:Npn \@@_hdottedline_i:n #1
  {
    \pgfrememberpicturepositiononpagetrue
    \@@_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).

```
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
    \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

The dotted line will be extended if the user uses `margin` (or `left-margin` and `right-margin`).

The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

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

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

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}
```

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

```
    \@@_qpoint:n { col - 1 }
    \dim_set:Nn \l_@@_x_initial_dim
      {
        \pgf@x +
        \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
        - \l_@@_left_margin_dim
      }
    \@@_qpoint:n { col - \@@_succ:n \c@jCol }
    \dim_set:Nn \l_@@_x_final_dim
      {
        \pgf@x -
        \bool_if:NTF \l_@@_NiceTabular_bool \tabcolsep \arraycolsep
        + \l_@@_right_margin_dim
      }
```

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.

```
    \tl_set:Nn \l_tmpa_tl { ( }
    \tl_if_eq:NNF \l_@@_left_delim_tl \l_tmpa_tl
      { \dim_gadd:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_inter_dots_dim } }
    \tl_set:Nn \l_tmpa_tl { ) }
    \tl_if_eq:NNF \l_@@_right_delim_tl \l_tmpa_tl
      { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
```

As for now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier "`:`" in the preamble. That's why we impose the style `standard`.

```
    \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
    \@@_draw_line:
  }
```

Vertical dotted lines

```
\cs_new_protected:Npn \@@_vdottedline:n #1
  {
    \bool_set_true:N \l_@@_initial_open_bool
    \bool_set_true:N \l_@@_final_open_bool
```

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

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

\cs_new_protected:Npn \@@_vdottedline_i:n #1
  {
```

The command `\CT@arc@` is a command of colortbl which sets the color of the rules in the array. The package nicematrix uses it even if colortbl is not loaded.

```
    \CT@arc@
    \pgfrememberpicturepositiononpagetrue
    \@@_qpoint:n { col - \int_eval:n { #1 + 1 } }
```

We do a translation par `-\l_@@_radius_dim` because we want the dotted line to have exactly the same position as a vertical rule drawn by "`|`" (considering the rule having a width equal to the diameter of the dots).

```
    \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }
    \dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }
    \@@_qpoint:n { row - 1 }
```

We arbitrary decrease the height of the dotted line by a quantity equal to `\l_@@_inter_dots_dim` in order to improve the visual impact.

```
    \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }
    \@@_qpoint:n { row - \@@_succ:n \c@iRow }
    \dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }
```

As for now, we have no option to control the style of the lines drawn by `\hdottedline` and the specifier "`:`" in the preamble. That's why we impose the style `standard`.

```
    \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
    \@@_draw_line:
  }
```

The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in "auto" mode.

```
\bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
\keys_define:nn { NiceMatrix / NiceMatrixBlock }
  {
    auto-columns-width .code:n =
      {
```

```
        \bool_set_true:N \l_@@_block_auto_columns_width_bool
        \dim_gzero_new:N \g_@@_max_cell_width_dim
        \bool_set_true:N \l_@@_auto_columns_width_bool
      }
  }

\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
  {
    \int_gincr:N \g_@@_NiceMatrixBlock_int
    \dim_zero:N \l_@@_columns_width_dim
    \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
          {
            \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
              { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
          }
      }
  }
```

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`).

```
  {
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \iow_shipout:Nn \@mainaux \ExplSyntaxOn
        \iow_shipout:Nx \@mainaux
          {
            \cs_gset:cpn
              { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

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

```
              { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
          }
        \iow_shipout:Nn \@mainaux \ExplSyntaxOff
      }
  }
```

The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```
\cs_generate_variant:Nn \dim_min:nn { v n }
\cs_generate_variant:Nn \dim_max:nn { v n }
```

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`.

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

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

```
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \int_step_variable:nnNn
          \l_@@_first_col_int \g_@@_col_total_int \@@_j:
```

If the cell (i-j) is empty or an implicit cell (that is to say a cell after implicit ampersands `&`) we don't update the dimensions we want to compute.

```
          {
            \cs_if_exist:cT
              { 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`.

```
              {
                \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
                \dim_set:cn { l_@@_row_\@@_i: _min_dim}
                  { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
                \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                  {
                    \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
                      { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
                  }
```

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

```
                \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
                \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
                  { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
                \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                  {
                    \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
                      { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
                  }
              }
          }
      }
```

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

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

```
        {
          \@@_qpoint:n { row - \@@_i: - base }
          \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
          \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
        }
    }
  \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
    {
      \dim_compare:nNnT
        { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
        {
          \@@_qpoint:n { col - \@@_j: }
          \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
          \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
        }
    }
}
```

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

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

Now, we can create the "medium nodes". We use a command `\@@_create_nodes:` because this command will also be used for the creation of the "large nodes".

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

The command `\@@_create_large_nodes:` must be used when we want to create only the "large nodes" and not the medium ones[40]. However, the computation of the mathematical coordinates of the "large nodes" needs the computation of the mathematical coordinates of the "medium nodes". Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```
\cs_new_protected:Npn \@@_create_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_computations_for_medium_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_computations_for_medium_nodes:
```

Now, we can create the "medium nodes". We use a command `\@@_create_nodes:` because this command will also be used for the creation of the "large nodes".

```
      \tl_set:Nn \l_@@_suffix_tl { - medium }
```

[40] If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```
      \@@_create_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
```

For “large nodes”, the exterior rows and columns don’t interfer. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```
\cs_new_protected:Npn \@@_computations_for_large_nodes:
  {
    \int_set:Nn \l_@@_first_row_int 1
    \int_set:Nn \l_@@_first_col_int 1
```

We have to change the values of all the dimensions `l_@@_row_`*i*`_min_dim`, `l_@@_row_`*i*`_max_dim`, `l_@@_column_`*j*`_min_dim` and `l_@@_column_`*j*`_max_dim`.

```
    \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
      {
        \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
          {
            (
              \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
              \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
          { l_@@_row_\@@_i: _min_dim }
      }
    \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
      {
        \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
          {
            (
              \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
              \dim_use:c
                { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
          { l_@@_column _ \@@_j: _ max _ dim }
      }
```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```
    \dim_sub:cn
      { l_@@_column _ 1 _ min _ dim }
      \l_@@_left_margin_dim
    \dim_add:cn
      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
      \l_@@_right_margin_dim
  }
```

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`).

```
\cs_new_protected:Npn \@@_create_nodes:
  {
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
```

```
      {
        \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
          {
```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```
            \@@_pgf_rect_node:nnnnn
              { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
              { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
            \str_if_empty:NF \l_@@_name_str
              {
                \pgfnodealias
                  { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
                  { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              }
          }
      }
```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{`n`}{...}{...}` with n>1 was issued and in `\g_@@_multicolumn_sizes_seq` the correspondant values of n.

```
    \seq_mapthread_function:NNN
      \g_@@_multicolumn_cells_seq
      \g_@@_multicolumn_sizes_seq
      \@@_node_for_multicolumn:nn
  }


\cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
  {
    \cs_set:Npn \@@_i: { #1 }
    \cs_set:Npn \@@_j: { #2 }
  }
```

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").

```
\cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
  {
    \@@_extract_coords_values: #1 \q_stop
    \@@_pgf_rect_node:nnnnn
      { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
      { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
      { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
          { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
      }
  }
```

Block matrices

The code in this section if for the construction of *block matrices*. It has no direct link with the environment `{NiceMatrixBlock}`.

The following command will be linked to \Block in the environments of nicematrix. We define it with \NewDocumentCommand of xparse because it has an optional argument between < and > (for TeX instructions put before the math mode of the label)

```
\NewDocumentCommand \@@_Block: { O { } m D < > { } m }
  { \@@_Block_i #2 \q_stop { #1 } { #3 } { #4 } }
```

The first mandatory argument of \@@_Block: has a special syntax. It must be of the form i-j where i and j are the size (in rows and columns) of the block.

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

Now, the arguments have been extracted: #1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of key-values, #4 are the tokens to put before the math mode and #5 is the label of the block.

```
\cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
  {

    \tl_set:Nx \l_tmpa_tl
      {
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \int_eval:n { \c@iRow + #1 - 1 } }
          { \int_eval:n { \c@jCol + #2 - 1 } }
      }
```

Now, \l_tmpa_tl constains a "object" corresponding to the position of the block whith four components surrounded by brackets:
{*imin*}{*jmin*}{*imax*}{*jmax*}.
We store this information in the sequence \g_@@_pos_of_blocks_seq.

```
    \seq_gput_left:NV \g_@@_pos_of_blocks_seq \l_tmpa_tl
```

We also store a complete description of the block in the sequence \g_@@_blocks_seq. Of course, the sequences \g_@@_pos_of_blocks_seq and \g_@@_blocks_seq are redundant, but it's for efficiency. In \g_@@_blocks_seq, each block is represented by an "objet" with six components:
{*imin*}{*jmin*}{*imax*}{*jmax*}{*options*}{*contents*}.

```
    \seq_gput_left:Nx \g_@@_blocks_seq
      {
        \l_tmpa_tl
        { #3 }
        \exp_not:n { { #4 \@@_math_toggle_token: #5 \@@_math_toggle_token: } }
      }
  }
```

The key tikz is for Tikz options used when the PGF node of the block is created.

```
\keys_define:nn { NiceMatrix / Block }
  {
    tikz .tl_set:N = \l_@@_tikz_tl ,
    tikz .value_required:n = true ,
  }
```

The command \@@_draw_blocks: will draw all the blocks. This command is used after the construction of the array.

```
\cs_new_protected:Npn \@@_draw_blocks:
  { \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iii:nnnnnn ##1 } }
%     \end{macrocode}
%
%    \begin{macrocode}
\cs_new_protected:Npn \@@_Block_iii:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

The group is for the keys.

```
    \group_begin:
    \keys_set:nn { NiceMatrix / Block } { #5 }
```

```
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn { #3 } > \c@iRow }
      { \int_compare_p:nNn { #4 } > \c@jCol }
      { \msg_error:nnnn { nicematrix } { Block~too~large } { #1 } { #2 } }
      {
```

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

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

The construction of the node corresponding to the merged cells.

```
        \pgfpicture
          \pgfrememberpicturepositiononpagetrue
          \pgf@relevantforpicturesizefalse
          \@@_qpoint:n { row - #1 }
          \dim_set_eq:NN \l_tmpa_dim \pgf@y
          \@@_qpoint:n { col - #2 }
          \dim_set_eq:NN \l_tmpb_dim \pgf@x
          \@@_qpoint:n { row - \@@_succ:n { #3 } }
          \dim_set_eq:NN \l_tmpc_dim \pgf@y
          \@@_qpoint:n { col - \@@_succ:n { #4 } }
          \dim_set_eq:NN \l_tmpd_dim \pgf@x
```

We construct the node for the block with the name `(#1-#2-block)`.
The function `\@@_pgf_rect_node:nnnnn` takes as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```
          \begin { pgfscope }
          \exp_args:Nx \pgfset { \l_@@_tikz_tl }
          \@@_pgf_rect_node:nnnnn
            { \@@_env: - #1 - #2 - block }
            \l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim
          \end { pgfscope }
```

If the creation of the "medium nodes" is required, we create a "medium node" for the block. The function `\@@_pgf_rect_node:nnnnn` takes as arguments the name of the node and two PGF points.

```
        \bool_if:NT \l_@@_medium_nodes_bool
          {
            \@@_pgf_rect_node:nnn
              { \@@_env: - #1 - #2 - block - medium }
              { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
              { \pgfpointanchor { \@@_env: - #3 - #4 - medium } { south~east } }
          }
```

Now, we will put the label of the block.

```
        \int_compare:nNnTF { #1 } = { #3 }
          {
```

If the block has only one row, we want the label of the block perfectly aligned on the baseline of the row. That's why we have constructed a `\pgfcoordinate` on the baseline of the row, in the first column of the array. Now, we retrieve the y-value of that node and we store it in `\l_tmpa_dim`.

```
            \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }
```

We retrieve (in `\pgf@x`) the x-value of the center of the block.

```
            \@@_qpoint:n { #1 - #2 - block }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
            \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
            \pgfnode { rectangle } { base }
              { \box_use_drop:N \l_@@_cell_box } { } { }
          }
```

If the number of rows is different of 1, we put the label of the block in the center of the node (the label of the block has been composed in `\l_@@_cell_box`).

```
          {
            \pgftransformshift { \@@_qpoint:n { #1 - #2 - block } }
            \pgfnode { rectangle } { center }
              { \box_use_drop:N \l_@@_cell_box } { } { }
          }
        \endpgfpicture
      }
    \group_end:
  }
```

How to draw the dotted lines transparently

```
\cs_set_protected:Npn \@@_renew_matrix:
  {
    \RenewDocumentEnvironment { pmatrix } { }
      { \pNiceMatrix }
      { \endpNiceMatrix }
    \RenewDocumentEnvironment { vmatrix } { }
      { \vNiceMatrix }
      { \endvNiceMatrix }
    \RenewDocumentEnvironment { Vmatrix } { }
      { \VNiceMatrix }
      { \endVNiceMatrix }
    \RenewDocumentEnvironment { bmatrix } { }
      { \bNiceMatrix }
      { \endbNiceMatrix }
    \RenewDocumentEnvironment { Bmatrix } { }
      { \BNiceMatrix }
      { \endBNiceMatrix }
  }
```

Automatic arrays

```
\cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
  {
    \int_set:Nn \l_@@_nb_rows_int { #1 }
    \int_set:Nn \l_@@_nb_cols_int { #2 }
  }
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
  {
    \int_zero_new:N \l_@@_nb_rows_int
    \int_zero_new:N \l_@@_nb_cols_int
    \@@_set_size:n #4 \q_stop
    \begin { NiceArrayWithDelims } { #1 } { #2 }
      { * { \l_@@_nb_cols_int } { C } } [ #3 , #5 , #7 ]
    \int_compare:nNnT \l_@@_first_row_int = 0
      {
        \int_compare:nNnT \l_@@_first_col_int = 0 { & }
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \prg_replicate:nn \l_@@_nb_rows_int
      {
        \int_compare:nNnT \l_@@_first_col_int = 0 { & }
```

You put `{ }` before `#6` to avoid a hasty expansion of an eventual `\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`).

```
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \int_compare:nNnT \l_@@_last_row_int > { -2 }
      {
        \int_compare:nNnT \l_@@_first_col_int = 0 { & }
        \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
        \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
      }
    \end { NiceArrayWithDelims }
  }
\cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
  {
    \cs_set_protected:cpn { #1 AutoNiceMatrix }
      {
        \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
        \AutoNiceMatrixWithDelims { #2 } { #3 }
      }
  }
\@@_define_com:nnn p ( )
\@@_define_com:nnn b [ ]
\@@_define_com:nnn v | |
\@@_define_com:nnn V \| \|
\@@_define_com:nnn B \{ \}
```

We define also an command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```
\NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
  {
    \group_begin:
      \bool_set_true:N \l_@@_NiceArray_bool
      \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
    \group_end:
  }
```

The redefinition of the command \dotfill

```
\cs_set_eq:NN \@@_dotfill \dotfill
\cs_new_protected:Npn \@@_dotfill:
  {
```

First, we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in case of use of `\dotfill` "internally" in the cell (e.g. `\hbox to 1cm {\dotfill}`).

```
    \@@_dotfill
    \bool_if:NT \l_@@_NiceTabular_bool
      { \group_insert_after:N \@@_dotfill_ii: }
      { \group_insert_after:N \@@_dotfill_i: }
  }
\cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
\cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }
```

Now, if the box if not empty (unfornately, we can't actually test whether the box is empty and that's why we only consider it's width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```
\cs_new_protected:Npn \@@_dotfill_iii:
  { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim  \@@_dotfill }
```

The command \diagbox

```
\cs_new_protected:Npn \@@_diagbox:nn #1 #2
  {
    \tl_gput_right:Nx \g_@@_internal_code_after_tl
      {
        \@@_actually_diagbox:nnnn
          { \int_use:N \c@iRow } { \int_use:N \c@jCol } { #1 } { #2 }
      }
```

```
  }
```

The two arguments of `\@@_actually_diagbox:nn` are the number of row and the number of column of the cell to slash. The two other are the elements to draw below and above the diagonal line.

```
\cs_new_protected:Npn \@@_actually_diagbox:nnnn #1 #2 #3 #4
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \pgfrememberpicturepositiononpagetrue
    \@@_qpoint:n { row - #1 }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - #2 }
    \dim_set_eq:NN \l_tmpb_dim \pgf@x
    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
    \@@_qpoint:n { row - \@@_succ:n { #1 } }
    \dim_set_eq:NN \l_tmpc_dim \pgf@y
    \@@_qpoint:n { col - \@@_succ:n { #2 } }
    \dim_set_eq:NN \l_tmpd_dim \pgf@x
    \pgfpathlineto { \pgfpoint \l_tmpd_dim \l_tmpc_dim }
    {
```

The command `\CT@arc@` is a command of colortbl which sets the color of the rules in the array. The package nicematrix uses it even if colortbl is not loaded.

```
      \CT@arc@
      \pgfsetroundcap
      \pgfusepathqstroke
    }
    \pgfset { inner~sep = 1 pt }
    \pgfscope
    \pgftransformshift { \pgfpoint \l_tmpb_dim \l_tmpc_dim }
    \pgfnode { rectangle } { south~west }
      { \@@_math_toggle_token: #3 \@@_math_toggle_token: } { } { }
    \endpgfscope
    \pgftransformshift { \pgfpoint \l_tmpd_dim \l_tmpa_dim }
    \pgfnode { rectangle } { north~east }
      { \@@_math_toggle_token: #4 \@@_math_toggle_token: } { } { }
    \endpgfpicture
  }
```

The command \CodeAfter

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the "normal syntax". For the case of "light-syntax", see the definition of the environment `{@@-light-syntax}` on p. 66.

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

```
\cs_new_protected:Npn \@@_CodeAfter:n #1 \end
  {
    \tl_gput_right:Nn \g_@@_code_after_tl { #1 }
    \@@_CodeAfter_i:n
  }
```

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

```
\cs_new_protected:Npn \@@_CodeAfter_i:n #1
  {
```

If this is really the end of the current environment (of nicematrix), we put back the command `\end` and its argument in the TeX flow.

```
    \str_set:NV \l_tmpa_str \@currenvir
    \bool_if:NTF { \str_if_eq_p:Vn \l_tmpa_str { #1 } }
      { \end { #1 } }
```

It this is not the `\end` we are looking for, we put those tokens in `\g_@@_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```
      {
```

```
        \tl_gput_right:Nn \g_@@_code_after_tl { \end { #1 } }
        \@@_CodeAfter:n
      }
  }
```

We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.
We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.
Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

```
\keys_define:nn { NiceMatrix / Package }
  {
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    renew-matrix .code:n = \@@_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    transparent .meta:n = { renew-dots , renew-matrix } ,
    transparent .value_forbidden:n = true,
  }
\ProcessKeysOptions { NiceMatrix / Package }
```

Error messages of the package

The following command converts all the elements of a sequence (which are token lists) into strings.

```
\cs_new_protected:Npn \@@_convert_to_str_seq:N #1
  {
    \seq_clear:N \l_tmpa_seq
    \seq_map_inline:Nn #1
      {
        \seq_put_left:Nx \l_tmpa_seq { \tl_to_str:n { ##1 } }
      }
    \seq_set_eq:NN #1 \l_tmpa_seq
  }
```

The following command creates a sequence of strings (`str`) from a `clist`.

```
\cs_new_protected:Npn \@@_set_seq_of_str_from_clist:Nn #1 #2
  {
    \seq_set_from_clist:Nn #1 { #2 }
    \@@_convert_to_str_seq:N #1
  }
\@@_set_seq_of_str_from_clist:Nn \c_@@_types_of_matrix_seq
  {
    NiceMatrix ,
    pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
  }
```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```
\cs_new_protected:Npn \@@_error_too_much_cols:
  {
    \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
      {
        \int_compare:nNnTF \l_@@_last_col_int = { -2 }
```

```
        { \@@_fatal:n { too~much~cols~for~matrix } }
        {
          \bool_if:NF \l_@@_last_col_without_value_bool
            { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
        }
      }
      { \@@_fatal:n { too~much~cols~for~array } }
  }
```

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

```
\cs_new:Npn \@@_message_hdotsfor:
  {
    \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
      { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
  }
\@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
    columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~potential~
    exterior~ones).~This~error~is~fatal.
  }
\@@_msg_new:nn { too~much~cols~for~matrix }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
    number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
    'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~
    This~error~is~fatal.
  }
```

For the following message, remind that the test is not done after the construction of the array but in each row. That's why we have to put `\c@jCol-1` and not `\c@jCol`.

```
\@@_msg_new:nn { too~much~cols~for~array }
  {
    You~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
    \int_eval:n { \c@jCol - 1 }~(plus~the~potential~exterior~ones).~
    This~error~is~fatal.
  }
\@@_msg_new:nn { in~first~col }
  {
    You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~last~col }
  {
    You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~first~row }
  {
    You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { in~last~row }
  {
    You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
    If~you~go~on,~this~command~will~be~ignored.
  }
```

```
\@@_msg_new:nn { option~S~without~siunitx }
  {
    You~can't~use~the~option~'S'~in~your~environment~\@@_full_name_env:
    because~you~have~not~loaded~siunitx.\\
    If~you~go~on,~this~option~will~be~ignored.
  }
\@@_msg_new:nn { bad~option~for~line-style }
  {
    Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
    is~'standard'.~If~you~go~on,~this~option~will~be~ignored.
  }
\@@_msg_new:nn { Unknown~option~for~xdots }
  {
    As~for~now~there~is~only~three~options~available~here:~'color',~'line-style'~
    and~'shorten'~(and~you~try~to~use~'\l_keys_key_str').~If~you~go~on,~
    this~option~will~be~ignored.
  }
\@@_msg_new:nn { ampersand~in~light-syntax }
  {
    You~can't~use~an~ampersand~(\token_to_str &)~to~separate~columns~because
    ~you~have~used~the~option~'light-syntax'.~This~error~is~fatal.
  }
\@@_msg_new:nn { double-backslash~in~light-syntax }
  {
    You~can't~use~\token_to_str:N \\~to~separate~rows~because~you~have~used~
    the~option~'light-syntax'.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
    (set~by~the~option~'end-of-row').~This~error~is~fatal.
  }
\@@_msg_new:nn { standard-cline~in~document }
  {
    The~key~'standard-cline'~is~available~only~in~the~preamble.\\
    If~you~go~on~this~command~will~be~ignored.
  }
\@@_msg_new:nn { bad~value~for~baseline }
  {
    The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
    valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
    \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
    If~you~go~on,~a~value~of~1~will~be~used.
  }
\@@_msg_new:nn { empty~environment }
  { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }
\@@_msg_new:nn { unknown~cell~for~line~in~code-after }
  {
    Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code-after'~
    can't~be~executed~because~a~cell~doesn't~exist.\\
    If~you~go~on~this~command~will~be~ignored.
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceArray }
  {
    In~the~\@@_full_name_env:,~you~must~use~the~option~
    'last-col'~without~value.\\
    However,~you~can~go~on~for~this~time~
    (the~value~'\l_keys_value_tl'~will~be~ignored).
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
  {
    In~\NiceMatrixoptions,~you~must~use~the~option~
    'last-col'~without~value.\\
```

```
    However,~you~can~go~on~for~this~time~
    (the~value~'\l_keys_value_tl'~will~be~ignored).
  }
\@@_msg_new:nn { Block~too~large }
  {
    You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
    too~small~for~that~block. \\
    If~you~go~on,~this~command~will~be~ignored.
  }
\@@_msg_new:nn { Wrong~last~row }
  {
    You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
    \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
    If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
    last~row.~You~can~avoid~this~problem~by~using~'last-row'~
    without~value~(more~compilations~might~be~necessary).
  }
\@@_msg_new:nn { Yet~in~env }
  {
    Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~nested.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { Outside~math~mode }
  {
    The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
    (and~not~in~\token_to_str:N \vcenter).\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { Bad~value~for~letter~for~dotted~lines }
  {
    The~value~of~key~'\tl_use:N\l_keys_key_str'~must~be~of~length~1.\\
    If~you~go~on,~it~will~be~ignored.
  }
\@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
  {
    The~key~'\tl_use:N\l_keys_key_str'~is~unknown~for~the~command~
    \token_to_str:N \NiceMatrixOptions. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    allow-duplicate-names,~
    code-for-first-col,~
    cell-space-bottom-limit,~
    cell-space-top-limit,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    end-of-row,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    letter-for-dotted-lines,~
```

```
    light-syntax,~
    nullify-dots,~
    renew-dots,~
    renew-matrix,~
    right-margin,~
    small,~
    transparent,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Unknown~option~for~NiceArray }
  {
    The~option~'\tl_use:N\l_keys_key_str'~is~unknown~for~the~environment~
    \{NiceArray\}. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~options,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    columns-width,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    nullify-dots,~
    renew-dots,~
    right-margin,~
    rules/color,~
    rules/width,~
    small,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
```

This error message is used for the set of keys `NiceMatrix/NiceMatrix` and `NiceMatrix/pNiceArray` (but not by `NiceMatrix/NiceArray` because, for this set of keys, there is also the options `t`, `c` and `b`).

```
\@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
  {
```

```
    The~option~'\tl_use:N\l_keys_key_str'~is~unknown~for~the~
    \@@_full_name_env:. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~options,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    columns-width,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    l~(=L),~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    nullify-dots,~
    r~(=R),~
    renew-dots,~
    right-margin,~
    rules/color,~
    rules/width,~
    S,~
    small,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Unknown~option~for~NiceTabular }
  {
    The~option~'\tl_use:N\l_keys_key_str'~is~unknown~for~the~environment~
    \{NiceTabular\}. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~*principal*~available~options,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col,~
```

```
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    columns-width,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    name,~
    nullify-dots,~
    renew-dots,~
    right-margin,~
    rules/color,~
    rules/width,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Duplicate~name }
  {
    The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
    the~same~environment~name~twice.~You~can~go~on,~but,~
    maybe,~you~will~have~incorrect~results~especially~
    if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
    message~again,~use~the~option~'allow-duplicate-names'.\\
    For~a~list~of~the~names~already~used,~type~H~<return>. \\
  }
  {
    The~names~already~defined~in~this~document~are:~
    \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
  }
\@@_msg_new:nn { Option~auto~for~columns-width }
  {
    You~can't~give~the~value~'auto'~to~the~option~'columns-width'~here.~
    If~you~go~on,~the~option~will~be~ignored.
  }
\@@_msg_new:nn { Zero~row }
  {
    There~is~a~problem.~Maybe~you~have~used~l,~c~and~r~instead~of~L,~C~
    and~R~in~the~preamble~of~your~environment. \\
    This~error~is~fatal.
  }
```

16 History

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 StackExchange[41], Tikz externalization is now deactivated in the environments of the package nicematrix.[42]

[41] cf. `tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package`

[42] Before this version, there was an error when using nicematrix with Tikz externalization. In any case, it's not possible to externalize the Tikz elements constructed by nicematrix because they use the options `overlay` and `remember picture`.

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the "main matrix" (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it's possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots\cdots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `O { }` in commands and environments defined with xparse by `! O { }` (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 `\hdashline` of arydshln).
Possibility to draw vertical dotted lines to separate columns with the specifier "`:`" in the preamble (similar to the classical specifier "`|`" and the specifier "`:`" of arydshln).

Changes between version 2.2 and 2.2.1

Improvment of the vertical dotted lines drawn by the specifier ":" in the preamble.
Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of siunitx.
Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of "`|`") as `\hdotsfor` does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.
Error message when the user gives an incorrect value for `last-row`.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol "`:`" (in the preamble of the array) and `\line` in `code-after`).
The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by "`|`") are now compatible with the color fixed by colortbl.
Correction of a bug: it was not possible to use the colon "`:`" in the preamble of an array when `pdflatex` was used with french-babel (because french-babel activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.
The option `columns-width=auto` doesn't need any more a second compilation.
The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).
The previous version of nicematrix was incompatible with a recent version of expl3 (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange[43], optimization of Tikz externalization is disabled in the environments of nicematrix when the class standalone or the package standalone is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

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

Changes between version 3.6 and 3.7

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

[43] cf. `tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize`

Changes between version 3.7 and 3.8

New programmation for the command `\Block` when the block has only one row. With this programmation, the vertical rules drawn by the specifier "`|`" at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`.
An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

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

Changes between version 3.9 and 3.10

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

Changes between versions 3.10 and 3.11

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

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.
Options `vlines`, `hlines` and `hvlines`.
Option `baseline` pour `{NiceArray}` (not for the other environments).
The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell i-j, the name is i-j`-block` and, if the creation of the "medium nodes" is required, a node i-j`-block-medium` is created.
If the user try to use more columns than allowed by its environment, an error is raised by nicematrix (instead of a low-level error).
The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the "last row".
The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customization of the dotted lines.
In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it's possible to use the options `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 command `\CodeAfter` (in the environments of nicematrix).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`
Commands to color cells, row and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`
New command `\diagbox`
The key `hvline` don't draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

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

A

B

Q

R

S

T

U

V

X

Contents