The package nicematrix[*]

F. Pantigny
fpantigny@wanadoo.fr

November 11, 2019

Abstract

The LaTeX package nicematrix provides new environments similar to the classical environments `{array}` and `{matrix}` but with some additional features. Among these features are the possibilities to fix the width of the columns and to draw continuous ellipsis dots between the cells of the array.

1 Presentation

This package can be used with `xelatex`, `lualatex`, `pdflatex` but also by the classical workflow `latex-dvips-ps2pdf` (or Adobe Distiller). Two or three compilations may be necessary. This package requires and **loads** the packages expl3, l3keys2e, xparse, array, amsmath and tikz. It also loads the Tikz library fit. The final user only has to load the extension with `\usepackage{nicematrix}`.

This package provides some new tools to draw mathematical matrices. The main features are the following:

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

- continuous dotted lines[1];
- exterior rows and columns for labels;
- a control of the width of the columns.

A command `\NiceMatrixOptions` is provided to fix the options (the scope of the options fixed by this command is the current TeX group).

An example for the continuous dotted lines

For example, consider the following code which uses an environment `{pmatrix}` of amsmath.

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

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

This code composes the matrix A on the right.

Now, if we use the package nicematrix with the option `transparent`, the same code will give the result on the right.

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

[*]This document corresponds to the version 3.7 of nicematrix, at the date of 2019/11/11.

[1]If the class option `draft` is used, these dotted lines will not be drawn for a faster compilation.

2 The environments of this extension

The extension nicematrix defines the following new environments.

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

By default, the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` behave almost exactly as the corresponding environments of amsmath: `{matrix}`, `{pmatrix}`, `{bmatrix}`, `{Bmatrix}`, `{vmatrix}` and `{Vmatrix}`.

The environment `{NiceArray}` is similar to the environment `{array}` of the package `{array}`. However, for technical reasons, in the preamble of the environment `{NiceArray}`, the user must use the letters `L`, `C` and `R` instead of `l`, `c` and `r`. It's possible to use the constructions `w{...}{...}`, `W{...}{...}`[2], `|`, `>{...}`, `<{...}`, `@{...}`, `!{...}` and `*{n}{...}` but the letters `p`, `m` and `b` should not be used. See p. 7 the section relating to `{NiceArray}`.

3 The continuous dotted lines

Inside the environments of the extension 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`.[3]

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[4] 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.

```
\begin{bNiceMatrix}
a_1      & \Cdots &        & & a_1 \\
\Vdots   & a_2    & \Cdots & & a_2 \\
         & \Vdots & \Ddots \\
\\
a_1      & a_2    &        & & a_n
\end{bNiceMatrix}
```

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

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}
```

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

[2] However, for the columns of type `w` and `W`, the cells are composed in math mode (in the environments of nicematrix) whereas in `{array}` of array, they are composed in text mode.

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

[4] The precise definition of a "non-empty cell" is given below (cf. p. 14).

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[5]).

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}
```

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.[6]

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}
```

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

3.1 The option nullify-dots

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

```
$A = \begin{pmatrix}
a_0 & b \\
a_1 &   \\
a_2 &   \\
a_3 &   \\
a_4 &   \\
a_5 & b
\end{pmatrix}$
```

$$A = \begin{pmatrix} a_0 & b \\ a_1 & \\ a_2 & \\ a_3 & \\ a_4 & \\ a_5 & b \end{pmatrix}$$

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

```
$B = \begin{pmatrix}
a_0 & b      \\
a_1 & \vdots \\
a_2 & \vdots \\
a_3 & \vdots \\
a_4 & \vdots \\
a_5 & b
\end{pmatrix}$
```

$$B = \begin{pmatrix} a_0 & b \\ a_1 & \vdots \\ a_2 & \vdots \\ a_3 & \vdots \\ a_4 & \vdots \\ a_5 & b \end{pmatrix}$$

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

[6]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. 10

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

```
$C = \begin{pNiceMatrix}
a_0 & b       \\
a_1 & \Vdots  \\
a_2 & \Vdots  \\
a_3 & \Vdots  \\
a_4 & \Vdots  \\
a_5 & b
\end{pNiceMatrix}$
```

$$C = \begin{pmatrix} a_0 & b \\ a_1 & \vdots \\ a_2 & \vdots \\ a_3 & \vdots \\ a_4 & \vdots \\ a_5 & b \end{pmatrix}$$

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

```
$D = \begin{pNiceMatrix}[nullify-dots]
a_0 & b      \\
a_1 & \Vdots \\
a_2 &        \\
a_3 &        \\
a_4 &        \\
a_5 & b
\end{pNiceMatrix}$
```

$$D = \begin{pmatrix} a_0 & b \\ a_1 & \vdots \\ a_2 & \vdots \\ a_3 & \vdots \\ a_4 & \vdots \\ a_5 & b \end{pmatrix}$$

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

There must be no space before the opening bracket ([) of the options of the environment.

3.2 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.

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

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

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

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

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

The command \hdotsfor of amsmath takes an optional argument (between square brackets) which is used for fine tuning of the space between two consecutive dots. For homogeneity, \Hdotsfor has also an optional argument but this argument is discarded silently.
Remark: Unlike the command \hdotsfor of amsmath, the command \Hdotsfor may be used when the extension colortbl is loaded (but you might have problem if you use \rowcolor on the same row as \Hdotsfor).

3.3 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`.[7]

- The option `renew-dots`

  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`[3] 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}
```

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

4 The Tikz nodes created by nicematrix

The package nicematrix creates a Tikz node for each cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix. However, the user may wish to use directly these nodes. It's possible. First, the user have to give a name to the array (with the key called `name`). Then, the nodes are accessible through the names "*name-i-j*" where *name* is the name given to the array and i and j the numbers of the row and the column of the considered cell.

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

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

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

In the following example, we have underlined all the nodes of the matrix.

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

In fact, the package nicematrix can create "extra nodes". These new nodes are created if the option `create-extra-nodes` is used. There are two series of extra nodes: the "medium nodes" and the "large nodes".

[7]The options `renew-dots`, `renew-matrix` and `transparent` can be fixed with the command `\NiceMatrixOptiosn` 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 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.[8]

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

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`.[9]

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

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.

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

In this case, if we want a control over the height of the rows, we can add a `\strut` in each row of the array.

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

We explain below how to fill the nodes created by nicematrix (cf. p. 19).

5 The code-after

The option `code-after` may be used to give some code that will be excuted after the construction of the matrix (and, hence, after the construction of all the Tikz nodes).

In the `code-after`, the Tikz nodes should be accessed by a name of the form i-j (without the prefix of the name of the environment).

Moreover, a special command, called `\line` is available to draw directly dotted lines between nodes.

```
$\begin{pNiceMatrix}[code-after = \line{1-1}{3-3}]
0 & 0 & 0 \\
0 &   & 0 \\
0 & 0 & 0
\end{pNiceMatrix}$
```

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

[8] There is no "large nodes" created in the exterior rows and columns (for these rows and columns, cf. p. 7).

[9] 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.

6 The environment {NiceArray}

The environment `{NiceArray}` is similar to the environment `{array}`. As for `{array}`, the mandatory argument is the preamble of the array. However, for technical reasons, in this preamble, the user must use the letters `L`, `C` and `R`[10] instead of `l`, `c` and `r`. It's possible to use the constructions `w{...}{...}`, `W{...}{...}`, `|`, `>{...}`, `<{...}`, `@{...}`, `!{...}` and `*{n}{...}` but the letters `p`, `m` and `b` should not be used.[11]

The environment `{NiceArray}` accepts the classical options `t`, `c` and `b` of `{array}` but also other options defined by `nicematrix` (`renew-dots`, `columns-width`, etc.).

An example with a linear system (we need `{NiceArray}` for the vertical rule):

```
$\left[\begin{NiceArray}{CCCC|C}
a_1     & ?       & \Cdots & ?       & ?     \\
0       &         & \Ddots & \Vdots  & \Vdots\\
\Vdots  & \Ddots & \Ddots & ? \\
0       & \Cdots & 0       & a_n     & ?
\end{NiceArray}\right]$
```

$$\left[\begin{array}{cccc|c} a_1 & ? & \cdots & ? & ? \\ 0 & & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & ? & \vdots \\ 0 & \cdots & 0 & a_n & ? \end{array}\right]$$

In fact, there is also variants for the environment `{NiceArray}`: `{pNiceArray}`, `{bNiceArray}`, `{BNiceArray}`, `{vNiceArray}` and `{VNiceArray}`.
In the following example, we use an environment `{pNiceArray}` (we don't use `{pNiceMatrix}` because we want to use the types `L` and `R` — in `{pNiceMatrix}`, all the columns are of type `C`).

```
$\begin{pNiceArray}{LCR}
a_{11}     & \Cdots & a_{1n} \\
a_{21}     &        & a_{2n} \\
\Vdots     &        & \Vdots \\
a_{n-1,1} & \Cdots & a_{n-1,n}
\end{pNiceArray}$
```

$$\left(\begin{array}{lcr} a_{11} & \cdots & a_{1n} \\ a_{21} & & a_{2n} \\ \vdots & & \vdots \\ a_{n-1,1} & \cdots & a_{n-1,n} \end{array}\right)$$

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 delimiters.

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

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

7 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 has the number 0 (and not 1). Idem for the potential first column. In general cases, one must specify the number of the last row and the number of the last column as values of `last-row` and `last-col`.

[10] The column types `L`, `C` and `R` are defined locally inside `{NiceArray}` with `\newcolumntype` of `array`. This definition overrides an eventual previous definition. In fact, the column types `w` and `W` are also redefined.

[11] In a command `\multicolumn`, one should also use the letters `L`, `C`, `R`.

```
$\begin{pNiceMatrix}[first-row,last-row=5,first-col,last-col=5]
    & C_1    & C_2    & C_3    & C_4    &      \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
L_2 & a_{21} & a_{22} & a_{23} & a_{24} & L_2 \\
L_3 & a_{31} & a_{32} & a_{33} & a_{34} & L_3 \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
    & C_1    & C_2    & C_3    & C_4    &
\end{pNiceMatrix}$
```

$$\begin{array}{cccccc} & C_1 & C_2 & C_3 & C_4 & \\ L_1 & \multicolumn{4}{c}{\multirow{4}{*}{$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$}} & L_1 \\ L_2 & & & & & L_2 \\ L_3 & & & & & L_3 \\ L_4 & & & & & L_4 \\ & C_1 & C_2 & C_3 & C_4 & \end{array}$$

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: the first column will be automatically (and necessarily) of type `R` and the last column will be automatically of type `L`.
- In an environment with an explicit preamble, the option `last-col` must be used *without* value: the number of columns will be automatically computed from the preamble of the array.
- For the potential last row, the option `last-row` may, in fact, be used without value. In this case, nicematrix computes, during the first compilation, the number of row of the array and writes that information in the `.aux` file for the second run. In the following example, the option `last-row` will be used without value.

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}}
$\begin{pNiceArray}{CC|CC}[first-row,last-row,first-col,last-col]
    & C_1    & C_2    & C_3    & C_4    &      \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
L_2 & a_{21} & a_{22} & a_{23} & a_{24} & L_2 \\
\hline
L_3 & a_{31} & a_{32} & a_{33} & a_{34} & L_3 \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
    & C_1    & C_2    & C_3    & C_4    &
\end{pNiceArray}$
```

$$\begin{array}{cccccc} & \color{red}C_1 & \color{red}C_2 & \color{red}C_3 & \color{red}C_4 & \\ \color{blue}L_1 & \multicolumn{4}{c}{\multirow{4}{*}{$\left(\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right)$}} & \color{magenta}L_1 \\ \color{blue}L_2 & & & & & \color{magenta}L_2 \\ \color{blue}L_3 & & & & & \color{magenta}L_3 \\ \color{blue}L_4 & & & & & \color{magenta}L_4 \\ & \color{green}C_1 & \color{green}C_2 & \color{green}C_3 & \color{green}C_4 & \end{array}$$

Remarks

- As shown in the previous example, an horizontal rule (drawn by `\hline`) doesn't extend in the exterior columns and a vertical rule (specified by a "`|`" in the preamble of the array) doesn't extend in the exterior rows.[12]
- Logically, the potential option `columns-width` (described p. 10) 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).

8 The dotted lines to separate rows or columns

In the environments of the extension 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 ofarydshln).

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

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

In the environments with an explicit preamble (like `{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)
```

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

These dotted lines do *not* extend in the potential exterior rows and columns.

```
$\begin{pNiceArray}{CCC:C}[
    first-row,last-col,
    code-for-first-row = \color{blue}\scriptstyle,
    code-for-last-col = \color{blue}\scriptstyle ]
C_1 & C_2 & C_3 & C_4 \\
1 & 2 & 3 & 4 & L_1 \\
5 & 6 & 7 & 8 & L_2 \\
9 & 10 & 11 & 12 & L_3 \\
\hdottedline
13 & 14 & 15 & 16 & L_4
\end{pNiceArray}$
```

$$\begin{array}{cc} \begin{array}{cccc} C_1 & C_2 & C_3 & C_4 \end{array} & \\ \left(\begin{array}{ccc:c} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ \hdashline 13 & 14 & 15 & 16 \end{array}\right) & \begin{array}{c} L_1 \\ L_2 \\ L_3 \\ L_4 \end{array} \end{array}$$

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`. For example, in this document, we have loaded the extension arydshln which uses the letter ":" to specify a vertical dashed line. Thus, by using `letter-for-dotted-lines`, we can use the vertical lines of both arydshln and nicematrix.

```
\NiceMatrixOptions{letter-for-dotted-lines = V}
\left(\begin{NiceArray}{C|C:CVC}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{NiceArray}\right)
```

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

[12]The latter is not true when the extension arydshln is loaded besides nicematrix. In fact, nicematrix and arydhsln are not totally compatible because arydshln redefines many internals of array. On another note, if one really wants a vertical rule running in the first and in the last row, he should use `!{\vline}` instead of `|` in the preamble of the array.

9 The width of the columns

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

```
$\left(\begin{NiceArray}{wc{1cm}CC}
1  & 12 & -123 \\
12 & 0  & 0    \\
4  & 1  & 2
\end{NiceArray}\right)$
```

$$\left(\begin{array}{ccc} 1 & 12 & -123 \\ 12 & 0 & 0 \\ 4 & 1 & 2 \end{array}\right)$$

In the environments of nicematrix, it's also possible to fix the width of all the columns of a matrix directly with the option 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 \arraycolsep) is not suppressed (of course, it's possible to suppress this space by setting \arraycolsep equal to 0 pt).

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.[13]

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

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

Without surprise, it's possible to fix the 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.[14]

```
\begin{NiceMatrixBlock}[auto-columns-width]
$\begin{pNiceMatrix}
a & b \\ c & d \\
\end{pNiceMatrix}
=
\begin{pNiceMatrix}
1   & 1245 \\ 345 & 2 \\
\end{pNiceMatrix}$
\end{NiceMatrixBlock}
```

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

Several compilations may be necessary to achieve the job.

[13] 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).

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

10 Block matrices

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.
The command `\Block` must be used in the upper leftmost cell of the array with two arguments. The first argument is the size of the block with the syntax i-j where i is the number of rows of the block and j its number of columns. The second argument is the content of the block (composed in math mode).

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

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

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 specificy some TeX code which will be inserted before the beginning of the math mode.

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

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

For technical reasons, you can't write `\Block{`i-j`}{<}`. But you can write `\Block{`i-j`}<>{<}` with the expected result.

11 The option small

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

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

$$\left[\begin{smallmatrix}1&-2&3&4&5\\0&3&2&1&2\\0&1&1&2&3\end{smallmatrix}\right]\begin{smallmatrix}\\ L_2\gets 2L_1-L_2\\ L_3\gets L_1+L_3\end{smallmatrix}$$

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 extension array) whereas the environment {smallmatrix} is constructed directly with an \halign of TeX.

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

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

12 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 col[15]. Of course, the user must not change the value of these counters which are used internally by nicematrix.

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

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

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

The extension nicematrix also provides commands in order to compose automatically matrices from a general pattern. These commands are \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).

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

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

13 The option hlines

You can add horizontal rules between rows in the environments of nicematrix with the usual command \hline. But, by convenience, the extension nicematrix also provides the option hlines. With this option, all the horizontal rules will be drawn (excepted, of course, the rule before the potential "first row" and the rule after the potential "last row").

[15]We recall that the first row (if it exists) has the number 0 and that the first col (if it exists) has also the number 0).

```
$\begin{NiceArray}{|*{4}{C|}}[hlines,first-row,first-col]
  & e & a & b & c \\
e & e & a & b & c \\
a & a & e & c & b \\
b & b & c & e & a \\
c & c & b & a & e
\end{NiceArray}$
```

$$\begin{array}{c|c|c|c|c|}
 & e & a & b & c \\ \hline
e & e & a & b & c \\ \hline
a & a & e & c & b \\ \hline
b & b & c & e & a \\ \hline
c & c & b & a & e \\ \hline
\end{array}$$

14 Utilisation 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.

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

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

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

15 Technical remarks

15.1 Intersections of dotted lines

Since the version 3.1 of nicematrix, the dotted lines created by `\Cdots`, `\Ldots`, `\Vdots`, etc. can't intersect.[16]

That means that a dotted line created by one these commands automatically stops when it arrives on a dotted line already drawn. Therefore, the order in which dotted lines are drawn is important. Here's that order (by design) : `\Hdotsfor`, `\Vdots`, `\Ddots`, `\Iddots`, `\Cdots` and `\Ldots`.

With this structure, it's possible to draw the following matrix.

```
$\begin{pNiceMatrix}[nullify-dots]
1 & 2 & 3 & \Cdots & n \\
1 & 2 & 3 & \Cdots & n \\
\Vdots & \Cdots & & \Hspace*{15mm} & \Vdots \\
& \Cdots & & &  \\
& \Cdots & & &  \\
& \Cdots & & &  \\
\end{pNiceMatrix}$
```

$$\begin{pmatrix}
1 & 2 & 3 & \cdots\cdots\cdots & n \\
1 & 2 & 3 & \cdots\cdots\cdots & n \\
\vdots & \cdots & \cdots & \cdots\cdots\cdots & \vdots \\
\vdots & \cdots & \cdots & \cdots\cdots\cdots & \vdots \\
\vdots & \cdots & \cdots & \cdots\cdots\cdots & \vdots \\
\vdots & \cdots & \cdots & \cdots\cdots\cdots & \vdots
\end{pmatrix}$$

15.2 Diagonal lines

By default, all the diagonal lines[17] 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:

[16] Of the contrary, dotted lines created by `\hdottedline`, the letter ":" in the preamble of the array and the command `\line` in the `code-after` can have intersections with other dotted lines.

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

Example with parallelization (default):

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

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

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

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

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

The same example without parallelization:

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

15.3 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 with contents `\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 less than 0.5 pt is empty.

- A cell which contains a command `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` or `\Iddots` is empty. We recall that these commands should be used alone in a cell.

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

15.4 The option exterior-arraycolsep

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea[18]. The environment `{matrix}` of amsmath and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of amsmath prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`. The extension nicematrix does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` of array (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of nicematrix are not affected).

15.5 The class option draft

The package nicematrix is rather slow when drawing the dotted lines (generated by `\Cdots`, `\Ldots`, `\Ddots`, etc. but also by `\hdottedline` or the specifier `:`).[19]
That's why, when the class option `draft` is used, the dotted lines are not drawn, for a faster compilation.

15.6 A technical problem with the argument of \\

For technical, reasons, if you use the optional argument of the command `\\`, the vertical space added will also be added to the "normal" node corresponding at the previous node.

```
\begin{pNiceMatrix}
a & \frac AB \\[2mm]
b & c
\end{pNiceMatrix}
```

$$\begin{pmatrix} a & \frac{A}{B} \\[2mm] b & c \end{pmatrix}$$

There are two solutions to solve this problem. The first solution is to use a TeX command to insert space between the rows.

```
\begin{pNiceMatrix}
a & \frac AB \\
\noalign{\kern2mm}
b & c
\end{pNiceMatrix}
```

$$\begin{pmatrix} a & \frac{A}{B} \\[2mm] b & c \end{pmatrix}$$

The other solution is to use the command `\multicolumn` in the previous cell.

```
\begin{pNiceMatrix}
a & \multicolumn1C{\frac AB} \\[2mm]
b & c
\end{pNiceMatrix}
```

$$\begin{pmatrix} a & \frac{A}{B} \\[2mm] b & c \end{pmatrix}$$

15.7 Obsolete environments

The version 3.0 of nicematrix has introduced the environment `{pNiceArray}` (and its variants) with the options `first-row`, `last-row`, `first-col` and `last-col`.
Consequently the following environments present in previous versions of nicematrix are deprecated:

- `{NiceArrayCwithDelims}` ;
- `{pNiceArrayC}`, `{bNiceArrayC}`, `{BNiceArrayC}`, `{vNiceArrayC}`, `{VNiceArrayC}` ;

[18] 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).* It's possible to suppress these spaces for a given environment `{array}` with a construction like `\begin{array}{@{}ccccc@{}}...\end{array}`.

[19] The main reason is that we want dotted lines with round dots (and not square dots) with the same space on both extremities of the lines. To achieve this goal, we have to construct our own system of dotted lines.

- `{NiceArrayRCwithDelims}` ;
- `{pNiceArrayRC}`, `{bNiceArrayRC}`, `{BNiceArrayRC}`, `{vNiceArrayRC}`, `{VNiceArrayRC}`.

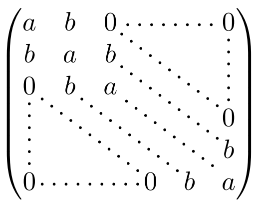
They might be deleted in a future version of nicematrix.

16 Examples

16.1 Dotted lines

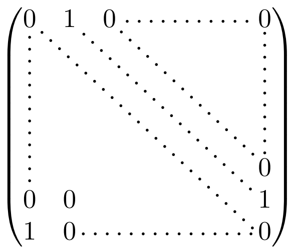
A tridiagonal matrix:

```
$\begin{pNiceMatrix}[nullify-dots]
a       & b       & 0       &         & \Cdots & 0       \\
b       & a       & b       & \Ddots &         & \Vdots \\
0       & b       & a       & \Ddots &         &         \\
        & \Ddots & \Ddots & \Ddots &         & 0       \\
\Vdots &         &         &         &         & b       \\
0       & \Cdots &         & 0       & b       & a
\end{pNiceMatrix}$
```



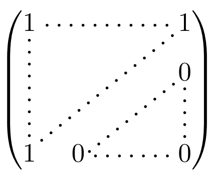
A permutation matrix:

```
$\begin{pNiceMatrix}
0       & 1 & 0 &         & \Cdots &   0    \\
\Vdots  &   &   & \Ddots &         & \Vdots \\
        &   &   & \Ddots &         &        \\
        &   &   & \Ddots &         &   0    \\
0       & 0 &   &         &         &   1    \\
1       & 0 &   & \Cdots &         &   0
\end{pNiceMatrix}$
```



An example with `\Iddots`:

```
$\begin{pNiceMatrix}
1       & \Cdots  &          & 1      \\
\Vdots  &         &          & 0      \\
        & \Iddots & \Iddots & \Vdots \\
1       & 0       & \Cdots  & 0
\end{pNiceMatrix}$
```



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}
```

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

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}
```

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 0\\
0 & 1 & 1 & 1 & 1 & 0\\
\vdots & \multicolumn{4}{c}{\cdots\cdots\cdots} & \vdots\\
\vdots & \multicolumn{4}{c}{\cdots\cdots\cdots} & \vdots\\
\vdots & \multicolumn{4}{c}{\cdots\cdots\cdots} & \vdots\\
\vdots & \multicolumn{4}{c}{\cdots\cdots\cdots} & \vdots\\
0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}$$

An example for the resultant of two polynoms:

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

$$\left|\begin{array}{cccc:ccc}
a_0 & & & & b_0 & & \\
a_1 & \ddots & & & b_1 & \ddots & \\
\vdots & \ddots & & & \vdots & \ddots & b_0\\
a_p & & & a_0 & & & b_1\\
 & \ddots & & a_1 & b_q & & \vdots\\
 & & & \vdots & & \ddots & \\
 & & & a_p & & & b_q
\end{array}\right|$$

An example for a linear system (the vertical rule has been drawn in cyan with the tools of colortbl):

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

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

16.2 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{code-for-last-col = \color{blue}\scriptstyle}
\setlength{\extrarowheight}{1mm}
\quad $\begin{pNiceArray}{CCCC:C}[last-col]
1&1&1&1&1&\\
2&4&8&16&9&\\
3&9&27&81&36&\\
4&16&64&256&100&
\end{pNiceArray}$
...
\end{NiceMatrixBlock}
```

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

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

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

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

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

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

16.3 How to highlight cells of the matrix

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").
In order to have the continuity of the lines, we have to set `inner sep = -\pgflinewidth/2`.

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

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

The package nicematrix is constructed upon the environment `{array}` and, therefore, it's possible to use the package colortbl in the environments of nicematrix. However, it's not always easy to do a fine tuning of colortbl. That's why we propose another 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`. Warning: some PDF readers are not able to render transparency correctly.

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

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

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

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
```

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

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

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

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

```
\tikzset{myoptions/.style={remember picture,
                           overlay,
                           name prefix = example-,
                           every node/.style = {fill = red!15,
                                                blend mode = multiply,
                                                inner sep = 0pt}}}
```

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

We obtain the following matrix.

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

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

```
\begin{pNiceArray}{>{\strut}CCCC}%
   [create-extra-nodes,margin,extra-margin=2pt,
    code-after = {\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) ; } ]
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{pNiceArray}
```

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

16.4 Direct utilisation of the Tikz nodes

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

The utilisation 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`.

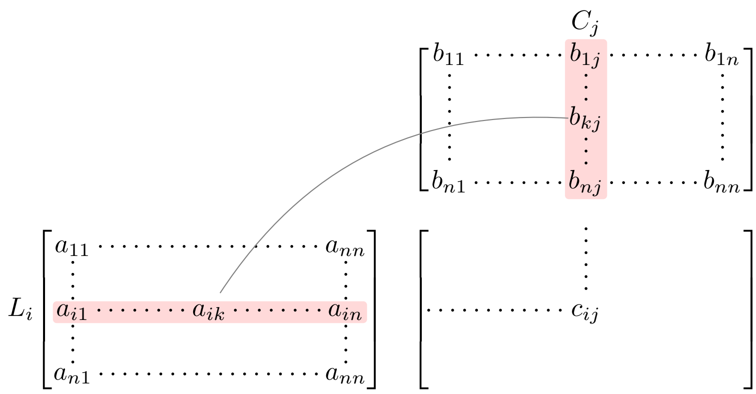
```
\begin{bNiceArray}{CC>{\strut}CCC}[name=A,first-col]
    & a_{11} & \Cdots &  &  & a_{nn} \\
    & \Vdots &       &  &  & \Vdots \\
L_i & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} \\
    & \Vdots &       &  &  & \Vdots \\
    & a_{n1} & \Cdots &  &  & a_{nn} \\
\end{bNiceArray}
&
```

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



17 Implementation

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

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\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`.

17.1 Declaration of the package and extensions loaded

First, tikz and the Tikz library `fit` are loaded before the `\ProvidesExplPackage`. They are loaded this way because `\usetikzlibrary` in `expl3` code fails.[20]

```
⟨@@=nm⟩
```

```
\RequirePackage{tikz}
\usetikzlibrary{fit}
\RequirePackage{expl3}[2019/07/01]
```

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

```
\RequirePackage{l3keys2e}
\ProvidesExplPackage
  {nicematrix}
  {\myfiledate}
  {\myfileversion}
  {Several features to improve the typesetting of mathematical matrices with TikZ}
```

We test if the class option `draft` has been used. In this case, we raise the flag `\c_@@_draft_bool` because we won't draw the dotted lines if the option `draft` is used.

```
\bool_new:N \c__nm_draft_bool
\DeclareOption { draft } { \bool_set_true:N \c__nm_draft_bool }
\DeclareOption* { }
\ProcessOptions \relax
```

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

We load array and amsmath.

```
\RequirePackage { array }
\RequirePackage { amsmath }
\RequirePackage { xparse } [ 2018-07-01 ]
```

```
\cs_new_protected:Npn \__nm_error:n { \msg_error:nn { nicematrix } }
\cs_new_protected:Npn \__nm_error:nn { \msg_error:nnn { nicematrix } }
\cs_new_protected:Npn \__nm_error:nnn { \msg_error:nnnn { nicematrix } }
\cs_new_protected:Npn \__nm_fatal:n { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \__nm_fatal:nn { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \__nm_msg_new:nn { \msg_new:nnn { nicematrix } }
\cs_new_protected:Npn \__nm_msg_new:nnn { \msg_new:nnnn { nicematrix } }
```

```
\cs_new_protected:Npn \__nm_msg_redirect_name:nn
  { \msg_redirect_name:nnn { nicematrix } }
```

17.2 Technical definitions

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__nm_revtex_bool
\@ifclassloaded { revtex4-1 }
  { \bool_set_true:N \c__nm_revtex_bool }
  { }
\@ifclassloaded { revtex4-2 }
  { \bool_set_true:N \c__nm_revtex_bool }
  { }
```

The following message must be defined right now because it may be used during the loading of the package.

```
\__nm_msg_new:nn { Draft~mode }
  { The~compilation~is~in~draft~mode:~the~dotted~lines~won't~be~drawn. }
```

[20] cf. `tex.stackexchange.com/questions/57424/using-of-usetikzlibrary-in-an-expl3-package-fails`

```
\bool_if:NT \c__nm_draft_bool
  { \msg_warning:nn { nicematrix }  { Draft~mode } }
```

We define a command `\iddots` similar to `\ddots` ($\ddots$) but with dots going forward (⋰). 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
      {
        \mkern 1 mu
        \raise \p@ \hbox:n { . }
        \mkern 2 mu
        \raise 4 \p@ \hbox:n { . }
        \mkern 2 mu
        \raise 7 \p@ \vbox { \kern 7 pt \hbox:n { . } } \mkern 1 mu
      }
  }
```

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

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__nm_env_int
```

We also define a counter to count the environments `{NiceMatrixBlock}`.

```
\int_new:N \g__nm_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__nm_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__nm_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__nm_in_env_bool
```

If the user uses `{NiceArray}` (and not another environment relying upon `{NiceArrayWithDelims}` like `{pNiceArray}`), we will raise the flag `\l_@@_NiceArray_bool`. We have to know that, because, in `{NiceArray}`, we won't use a structure with `\left` and `\right` and we will use the option of position (`t`, `b` or `c`).

```
\bool_new:N \l__nm_NiceArray_bool
```

```
\cs_new_protected:Npn \__nm_test_if_math_mode:
  {
    \if_mode_math: \else:
      \__nm_fatal:n { Outside~math~mode }
    \fi:
  }
```

Consider the following code:

```
$\begin{pNiceMatrix}
a & b & c \\
d & e & \Vdots \\
f & \Cdots \\
g & h & i \\
\end{pNiceMatrix}$
```

First, the dotted line created by the `\Vdots` will be drawn. The implicit cell in position 2-3 will be considered as “dotted”. Then, we will have to draw the dotted line specified by the `\Cdots`; the final extremity of that line will be exactly in position 2-3 and, for that new second line, it should be considered as a *closed* extremity (since it is dotted). However, we don’t have the (normal) Tikz node of that node (since it’s an implicit cell): we can’t draw such a line. That’s why that dotted line will be said *impossible* and an error will be raised.[21]

```
\bool_new:N \l__nm_impossible_line_bool
```

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

```
\bool_new:N \c__nm_colortbl_loaded_bool
\AtBeginDocument
  {
    \@ifpackageloaded { colortbl }
      {
        \bool_set_true:N \c__nm_colortbl_loaded_bool
        \cs_set_protected:Npn \__nm_vline_i: { { \CT@arc@ \vline } }
      }
      { }
  }
```

The length `\l_@@_inter_dots_dim` is the distance between two dots for the dotted lines. The default value is 0.45 em but it will be changed if the option `small` is used.

```
\dim_new:N \l__nm_inter_dots_dim
\dim_set:Nn \l__nm_inter_dots_dim { 0.45 em }
```

The length `\l_@@_radius_dim` is the radius of the dots for the dotted lines. The default value is 0.34 pt but it will be changed if the option `small` is used.

```
\dim_new:N \l__nm_radius_dim
\dim_set:Nn \l__nm_radius_dim { 0.53 pt }
```

The name of the current environment or the current command (will be used only in the error messages).

```
\str_new:N \g__nm_type_env_str
```

```
\tl_new:N \g__nm_code_after_tl
```

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

```
\int_new:N \l__nm_save_iRow_int
\int_new:N \l__nm_save_jCol_int
```

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

[21]Of course, the user should solve the problem by adding the lacking ampersands.

17.2.1 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. As usual, the global version is for the passage in the `\group_insert_after:N`.

```
\int_new:N \l__nm_first_row_int
\int_set:Nn \l__nm_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__nm_first_col_int
\int_set:Nn \l__nm_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__nm_last_row_int
\int_set:Nn \l__nm_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".[22]

```
\bool_new:N \l__nm_last_row_without_value_bool
```

- **Last column**

  For the eventual "last column", we use an integer. A value of −1 means that there is no last column.

```
\int_new:N \l__nm_last_col_int
\int_set:Nn \l__nm_last_col_int { -1 }
```

  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__nm_last_col_found_bool
```

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

[22]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.

17.2.2 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__nm_siunitx_loaded_bool
\AtBeginDocument
  {
    \@ifpackageloaded { siunitx }
      { \bool_set_true:N \c__nm_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 utilisation of an environment of nicematrix with the command `\@@_adapt_S_column:`.

```
\cs_set_protected:Npn \__nm_adapt_S_column:
  {
```

In the preamble of the LaTeX document, the boolean `\c_@@_siunitx_loaded_bool` won't be known. That's why we test the existence of `\c_@@_siunitx_loaded_bool` and not its value.[23]

```
    \bool_if:NT \c__nm_siunitx_loaded_bool
      {
        \group_begin:
        \@temptokena = { }
```

[23]Indeed, nicematrix may be used in the preamble of the LaTeX document. For example, in this document, we compose a matrix in the box `\ExampleOne` before loading arydshln (because arydshln is not totally compatible with nicematrix).

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__nm_table_collect_begin_tl
        \tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
        \tl_gset:Nx \c__nm_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
        \tl_new:N \c__nm_table_print_tl
        \tl_gset:Nx \c__nm_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 \__nm_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 \__nm_renew_NC@rewrite@S:
  {
    \renewcommand*{\NC@rewrite@S}[1][]
      {
        \@temptokena \exp_after:wN
          {
            \tex_the:D \@temptokena
            > { \__nm_Cell: \c__nm_table_collect_begin_tl S {##1} }
            c
            < { \c__nm_table_print_tl \__nm_end_Cell: }
          }
        \NC@find
      }
  }
```

17.3 The options

The token list \l_@@_pos_env_str will contain one of the three values t, c or b and will indicate the position of the environment as in the option of the environment {array}. For the environment {pNiceMatrix}, {pNiceArray} and their variants, the value will programmatically be fixed to c. For the environment {NiceArray}, however, the three values t, c and b are possible.

```
\str_new:N \l__nm_pos_env_str
\str_set:Nn \l__nm_pos_env_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__nm_exterior_arraycolsep_bool
```

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

```
\bool_new:N \l__nm_parallelize_diags_bool
\bool_set_true:N \l__nm_parallelize_diags_bool
```

The flag \l_@@_hlines_bool corresponds to the option \hlines.

```
\bool_new:N \l__nm_hlines_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__nm_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__nm_auto_columns_width_bool
```

The token list \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__nm_name_str
```

The boolean \l_@@_extra_nodes_bool will be used to indicate whether the "medium nodes" and "large nodes" are created in the array.

```
\bool_new:N \l__nm_extra_nodes_bool
\bool_new:N \g__nm_extra_nodes_bool
```

The dimensions \l_@@_left_margin_dim and \l_@@_right_margin_dim correspond to the options left-margin and right-margin.

```
\dim_new:N \l__nm_left_margin_dim
\dim_new:N \l__nm_right_margin_dim
\dim_new:N \g__nm_width_last_col_dim
\dim_new:N \g__nm_width_first_col_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__nm_extra_left_margin_dim
\dim_new:N \l__nm_extra_right_margin_dim
```

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

```
\keys_define:nn { NiceMatrix / Global }
  {
    code-for-first-col .tl_set:N = \l__nm_code_for_first_col_tl ,
    code-for-first-col .value_required:n = true ,
    code-for-last-col .tl_set:N = \l__nm_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    code-for-first-row .tl_set:N = \l__nm_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
    code-for-last-row .tl_set:N = \l__nm_code_for_last_row_tl ,
    code-for-last-row .value_required:n = true ,
    small .bool_set:N = \l__nm_small_bool ,
    hlines .bool_set:N = \l__nm_hlines_bool ,
    parallelize-diags .bool_set:N = \l__nm_parallelize_diags_bool ,
```

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

```
    renew-dots .bool_set:N = \l__nm_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    nullify-dots .bool_set:N = \l__nm_nullify_dots_bool ,
```

An option to test whether the extra nodes will be created (these nodes are the “medium nodes” and the “large nodes”). In some circonstancies, the extra nodes are created automatically, for example when a dotted line has an “open” extremity.

```
    create-extra-nodes .bool_set:N = \l__nm_extra_nodes_bool ,
    left-margin .dim_set:N = \l__nm_left_margin_dim ,
    left-margin .default:n = \arraycolsep ,
    right-margin .dim_set:N = \l__nm_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__nm_extra_left_margin_dim ,
    extra-right-margin .dim_set:N = \l__nm_extra_right_margin_dim ,
    extra-margin .meta:n =
     { extra-left-margin = #1 , extra-right-margin = #1 } ,
  }
```

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

```
\keys_define:nn { NiceMatrix / Env }
  {
    columns-width .code:n =
      \str_if_eq:nnTF { #1 } { auto }
        { \bool_set_true:N \l__nm_auto_columns_width_bool }
        { \dim_set:Nn \l__nm_columns_width_dim { #1 } } ,
    columns-width .value_required:n = true ,
    name .code:n =
      \unless \ifmeasuring@
         \str_set:Nn \l_tmpa_str { #1 }
         \seq_if_in:NVTF \g__nm_names_seq \l_tmpa_str
           { \__nm_error:nn { Duplicate~name } { #1 } }
           { \seq_gput_left:NV \g__nm_names_seq \l_tmpa_str }
         \str_set_eq:NN \l__nm_name_str \l_tmpa_str
      \fi ,
    name .value_required:n = true ,
    code-after .tl_gset:N = \g__nm_code_after_tl ,
    code-after .value_required:n = true ,
    first-col .code:n = \int_zero:N \l__nm_first_col_int ,
    first-row .code:n = \int_zero:N \l__nm_first_row_int ,
    last-row .int_set:N = \l__nm_last_row_int ,
    last-row .default:n = -1 ,
  }
```

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

```
        NiceMatrix / Global ,
        NiceMatrix / Env ,
      }
  }
```

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

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

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

```
    renew-matrix .code:n = \__nm_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    RenewMatrix .code:n = \__nm_error:n { Option~RenewMatrix~suppressed }
                          \__nm_renew_matrix: ,
    transparent .meta:n = { renew-dots , renew-matrix } ,
    transparent .value_forbidden:n = true,
    Transparent .code:n = \__nm_error:n { Option~Transparent~suppressed }
                          \__nm_renew_matrix:
                          \bool_set_true:N \l__nm_renew_dots_bool ,
```

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__nm_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 }
        { \__nm_error:n { Option~auto~for~columns-width } }
        { \dim_set:Nn \l__nm_columns_width_dim { #1 } } ,
```

Usually, an error is raised when the user tries to give the same to name 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 =
      \__nm_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 } = \c_one_int }
          { \str_set:Nx \l__nm_letter_for_dotted_lines_str { #1 } }
          { \__nm_error:n { Bad~value~for~letter~for~dotted~lines } }
      } ,
    letter-for-dotted-lines .value_required:n = true ,


    unknown .code:n  = \__nm_error:n { Unknown~key~for~NiceMatrixOptions }
  }
\str_new:N \l__nm_letter_for_dotted_lines_str
\str_set_eq:NN \l__nm_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}
                         { \__nm_error:n { last-col~empty~for~NiceMatrix } }
                         { \int_set:Nn \l__nm_last_col_int { #1 } } ,
    unknown .code:n = \__nm_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 }
  {
```

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__nm_pos_env_str c ,
    t .code:n = \str_set:Nn \l__nm_pos_env_str t ,
    b .code:n = \str_set:Nn \l__nm_pos_env_str b ,
```

In the environments `{NiceArray}` and its variants, the option `last-col` must be used without value because the number of columns of the array can be read in the preamble of the array.

```
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \__nm_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l__nm_last_col_int ,
    unknown .code:n = \__nm_error:n { Unknown~option~for~NiceArray }
  }
\keys_define:nn { NiceMatrix / pNiceArray }
  {
    first-col .code:n = \int_zero:N \l__nm_first_col_int ,
    last-col .code:n = \tl_if_empty:nF {#1}
                         { \__nm_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l__nm_last_col_int ,
    first-row .code:n = \int_zero:N \l__nm_first_row_int ,
    last-row .int_set:N = \l__nm_last_row_int ,
    last-row .default:n = -1 ,
    unknown .code:n = \__nm_error:n { Unknown~option~for~NiceMatrix }
  }
```

17.4 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:Nn \__nm_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 = \c_one_int
      {
        \int_compare:nNnT \l__nm_first_col_int = \c_one_int
          \__nm_begin_of_row:
      }
    \int_gset:Nn \g__nm_col_total_int
      { \int_max:nn \g__nm_col_total_int \c@jCol }
```

The content of the cell is composed in the box `\l_tmpa_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 `\c_math_toggle_token` also).

```
    \hbox_set:Nw \l_tmpa_box
    \c_math_toggle_token
    \bool_if:NT \l__nm_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). In a corner of the matrix, it would be logical to use none of the codes `\l_@@_code_for_first_row_tl` and *al.* As for now, this result is acheived only for the north-west corner (this allows an automatic numerotation of the rows ans the columns with `iRow` and `jCol` with the first col and the first row — probably the preferential choice for such a numerotation).

```
    \int_compare:nNnTF \c@iRow = \c_zero_int
      { \int_compare:nNnT \c@jCol > \c_zero_int \l__nm_code_for_first_row_tl }
      {
        \int_compare:nNnT \c@iRow = \l__nm_last_row_int
          \l__nm_code_for_last_row_tl
      }
  }
```

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

```
\cs_new_protected:Nn \__nm_begin_of_row:
  {
    \int_gincr:N \c@iRow
    \dim_gset_eq:NN \g__nm_dp_ante_last_row_dim \g__nm_dp_last_row_dim
    \dim_gzero:N \g__nm_dp_last_row_dim
    \dim_gzero:N \g__nm_ht_last_row_dim
  }
```

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.

```
\cs_new_protected:Npn \__nm_actualization_for_first_and_last_row:
  {
    \int_compare:nNnT \c@iRow = \c_zero_int
      {
        \dim_gset:Nn \g__nm_dp_row_zero_dim
          { \dim_max:nn \g__nm_dp_row_zero_dim { \box_dp:N \l_tmpa_box } }
        \dim_gset:Nn \g__nm_ht_row_zero_dim
          { \dim_max:nn \g__nm_ht_row_zero_dim { \box_ht:N \l_tmpa_box } }
      }
    \int_compare:nNnT \c@iRow = \c_one_int
      {
        \dim_gset:Nn \g__nm_ht_row_one_dim
          { \dim_max:nn \g__nm_ht_row_one_dim { \box_ht:N \l_tmpa_box } }
      }
    \dim_gset:Nn \g__nm_ht_last_row_dim
      { \dim_max:nn \g__nm_ht_last_row_dim { \box_ht:N \l_tmpa_box } }
```

```
    \dim_gset:Nn \g__nm_dp_last_row_dim
      { \dim_max:nn \g__nm_dp_last_row_dim { \box_dp:N \l_tmpa_box } }
  }
```

```
\cs_new_protected:Nn \__nm_end_Cell:
  {
    \c_math_toggle_token
    \hbox_set_end:
```

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__nm_max_cell_width_dim
      { \dim_max:nn \g__nm_max_cell_width_dim { \box_wd:N \l_tmpa_box } }
```

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

```
    \__nm_actualization_for_first_and_last_row:
```

Now, we can create the Tikz node of the cell.

```
    \tikz
      [
        remember~picture ,
        inner~sep = \c_zero_dim ,
        minimum~width = \c_zero_dim ,
        baseline
      ]
    \node
      [
        anchor = base ,
        name = nm - \int_use:N \g__nm_env_int -
                    \int_use:N \c@iRow -
                    \int_use:N \c@jCol ,
        alias =
          \str_if_empty:NF \l__nm_name_str
            {
              \l__nm_name_str -
              \int_use:N \c@iRow -
              \int_use:N \c@jCol
            }
      ]
    \bgroup
    \box_use:N \l_tmpa_box
    \egroup ;
  }
```

```
\cs_generate_variant:Nn \dim_set:Nn { N x }
```

In the environment `{NiceArrayWithDelims}`, we will have to redefine the column types `w` and `W`. These definitions are rather long because we have to construct the `w`-nodes in these columns. The redefinition of these two column types are very close and that’s why we use a macro `\@@_renewcolumntype:nn`. The first argument is the type of the column (`w` or `W`) and the second argument is a code inserted at a special place and which is the only difference between the two definitions.

```
\cs_new_protected:Nn \__nm_renewcolumntype:nn
  {
    \newcolumntype #1 [ 2 ]
      {
        > {
            \hbox_set:Nw \l_tmpa_box
            \__nm_Cell:
          }
        c
        < {
            \__nm_end_Cell:
            \hbox_set_end:
            #2
```

```
            \hbox_set:Nn \l_tmpb_box
              { \makebox [ ##2 ] [ ##1 ] { \box_use:N \l_tmpa_box } }
            \dim_set:Nn \l_tmpa_dim { \box_dp:N \l_tmpb_box }
            \box_move_down:nn \l_tmpa_dim
              {
                \vbox:n
                  {
                    \hbox_to_wd:nn { \box_wd:N \l_tmpb_box }
                      {
                        \hfil
                        \tikz [ remember~picture , overlay ]
                        \coordinate (__nm~north~east) ;
                      }
                    \hbox:n
                      {
                        \tikz [ remember~picture , overlay ]
                        \coordinate (__nm~south~west) ;
                        \box_move_up:nn \l_tmpa_dim { \box_use:N \l_tmpb_box }
                      }
                  }
              }
```

The `w`-node is created using the Tikz library `fit` after construction of the nodes `(@@~south~west)` and `(@@~north~east)`. It's not possible to construct by a standard `node` instruction because such a construction give an erroneous result with some engines (`XeTeX`, `LuaTeX`) although the result is good with `pdflatex` (why?).

```
            \tikz [ remember~picture , overlay ]
            \node
              [
                node~contents = { } ,
                name = nm - \int_use:N \g__nm_env_int -
                            \int_use:N \c@iRow -
                            \int_use:N \c@jCol - w,
                alias =
                  \str_if_empty:NF \l__nm_name_str
                    {
                      \l__nm_name_str -
                      \int_use:N \c@iRow -
                      \int_use:N \c@jCol - w
                    } ,
                inner~sep = \c_zero_dim ,
                fit = (__nm~south~west) (__nm~north~east)
              ]
            ;
          }
      }
  }
```

The argument of the following command `\@@_instruction_of_type:n` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). This command writes in the corresponding `\g_@@_`*type*`_lines_tl` the instruction which will really 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 \\
\end{pNiceMatrix}
```

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

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

```
\@@_draw_Cdots:nn {2}{2}
\@@_draw_Cdots:nn {3}{2}
```

We begin with a test of the flag `\c_@@_draft_bool` because, if the key `draft` is used, the dotted lines are not drawn.

```
\bool_if:NTF \c__nm_draft_bool
  { \cs_set_protected:Npn \__nm_instruction_of_type:n #1 { } }
  {
    \cs_new_protected:Npn \__nm_instruction_of_type:n #1
      {
```

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__nm_ #1 _ lines _ tl }
          {
            \use:c { __nm _ draw _ #1 : nn }
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
          }
      }
  }
```

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 \__nm_array:
  {
    \bool_if:NTF \c__nm_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_@@_pos_env_str` may have the value `t`, `c` or `b`.

```
    [ \l__nm_pos_env_str ]
  }
```

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

```
\cs_new:Npn \__nm_everycr:
  { \noalign { \__nm_everycr_i: } }
\cs_new_protected:Npn \__nm_everycr_i:
  {
    \int_gzero:N \c@jCol
    \bool_if:NT \l__nm_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__nm_last_row_int
              {
                \hrule \@height \arrayrulewidth
                \skip_vertical:n { - \arrayrulewidth }
              }
          }
      }
  }
```

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

```
\cs_new_protected:Npn \__nm_pre_array:
  {
    \cs_if_exist:NT \theiRow
      { \int_set_eq:NN \l__nm_save_iRow_int \c@iRow }
    \int_gzero_new:N \c@iRow
    \cs_if_exist:NT \thejCol
      { \int_set_eq:NN \l__nm_save_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__nm_small_bool
      {
        \cs_set:Npn \arraystretch { 0.47 }
        \dim_set:Nn \arraycolsep { 1.45 pt }
      }
```

We switch to a global version of the boolean `\g_@@_extra_nodes_bool`, because, in some circonstancies, the boolean will be raised from inside a cell of the `\halign` (in particular in a column of type `w`).

```
    \bool_gset_eq:NN \g__nm_extra_nodes_bool \l__nm_extra_nodes_bool
```

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:NTF \c__nm_colortbl_loaded_bool
          {
            \CT@everycr
              {
                \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
                \__nm_everycr:
              }
          }
          { \everycr { \__nm_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`[24] 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__nm_dp_row_zero_dim
        \dim_gset:Nn \g__nm_dp_row_zero_dim { \box_dp:N \@arstrutbox }
        \dim_gzero_new:N \g__nm_ht_row_zero_dim
        \dim_gset:Nn \g__nm_ht_row_zero_dim { \box_ht:N \@arstrutbox }
        \dim_gzero_new:N \g__nm_ht_row_one_dim
        \dim_gset:Nn \g__nm_ht_row_one_dim { \box_ht:N \@arstrutbox }
        \dim_gzero_new:N \g__nm_dp_ante_last_row_dim
        \dim_gset:Nn \g__nm_dp_ante_last_row_dim { \box_dp:N \@arstrutbox }
        \dim_gzero_new:N \g__nm_ht_last_row_dim
        \dim_gset:Nn \g__nm_ht_last_row_dim { \box_ht:N \@arstrutbox }
        \dim_gzero_new:N \g__nm_dp_last_row_dim
        \dim_gset:Nn \g__nm_dp_last_row_dim { \box_dp:N \@arstrutbox }
```

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

After its first utilisation, 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`.[25]

```
        \cs_set:Npn \ialign
          {
            \everycr { }
            \tabskip = \c_zero_skip
            \halign
          }
        \halign
      }
```

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}`.

```
    \newcolumntype L { > \__nm_Cell: l < \__nm_end_Cell: }
    \newcolumntype C { > \__nm_Cell: c < \__nm_end_Cell: }
    \newcolumntype R { > \__nm_Cell: r < \__nm_end_Cell: }
```

```
    \cs_set_eq:NN \Ldots \__nm_Ldots
    \cs_set_eq:NN \Cdots \__nm_Cdots
    \cs_set_eq:NN \Vdots \__nm_Vdots
    \cs_set_eq:NN \Ddots \__nm_Ddots
    \cs_set_eq:NN \Iddots \__nm_Iddots
    \cs_set_eq:NN \hdottedline \__nm_hdottedline:
    \cs_set_eq:NN \Hspace \__nm_Hspace:
    \cs_set_eq:NN \Hdotsfor \__nm_Hdotsfor:
    \cs_set_eq:NN \multicolumn \__nm_multicolumn:nnn
    \cs_set_eq:NN \Block \__nm_Block:
    \bool_if:NT \l__nm_renew_dots_bool
      {
        \cs_set_eq:NN \ldots \__nm_Ldots
        \cs_set_eq:NN \cdots \__nm_Cdots
        \cs_set_eq:NN \vdots \__nm_Vdots
        \cs_set_eq:NN \ddots \__nm_Ddots
        \cs_set_eq:NN \iddots \__nm_Iddots
        \cs_set_eq:NN \dots \__nm_Ldots
        \cs_set_eq:NN \hdotsfor \__nm_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__nm_multicolumn_cells_seq
    \seq_gclear_new:N \g__nm_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__nm_first_row_int - 1 }
```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows and `\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__nm_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__nm_col_total_int
```

```
    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

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

We nullify the definitions of the column types `w` and `W` before their redefinition because we want to avoid a warning in the log file for a redefinition of a column type. We must put `\relax` and not `\prg_do_nothing:`.

```
    \cs_set_eq:NN \NC@find@w \relax
    \cs_set_eq:NN \NC@find@W \relax
    \__nm_renewcolumntype:nn w { }
    \__nm_renewcolumntype:nn W { \cs_set_eq:NN \hss \hfil }
```

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 extensions, 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__nm_letter_for_dotted_lines_str { } \l__nm_letter_for_dotted_lines_str
    \exp_args:NV \newcolumntype \l__nm_letter_for_dotted_lines_str
      {
        !
          {
            \skip_horizontal:n { 0.53 pt }
```

If the array is an array with all the columns of the same width, we don't ask for the creation of the extra nodes because we will use the "`col`" nodes for the vertical dotted line.

```
            \bool_if:nF
              {
                \l__nm_auto_columns_width_bool
                || \dim_compare_p:nNn \l__nm_columns_width_dim > \c_zero_dim
              }
              { \bool_gset_true:N \g__nm_extra_nodes_bool }
```

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__nm_last_vdotted_col_int
              {
                \int_gset_eq:NN \g__nm_last_vdotted_col_int \c@jCol
                \tl_gput_right:Nx \g__nm_code_after_tl
```

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

```
                  { \__nm_vdottedline:n { \int_use:N \c@jCol } }
              }
          }
      }
    \int_gzero_new:N \g__nm_last_vdotted_col_int
    \bool_if:NT \c__nm_siunitx_loaded_bool \__nm_renew_NC@rewrite@S:
    \int_gset:Nn \g__nm_last_vdotted_col_int { -1 }
    \bool_gset_false:N \g__nm_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__nm_Cdots_lines_tl
    \tl_gclear_new:N \g__nm_Ldots_lines_tl
    \tl_gclear_new:N \g__nm_Vdots_lines_tl
    \tl_gclear_new:N \g__nm_Ddots_lines_tl
    \tl_gclear_new:N \g__nm_Iddots_lines_tl
    \tl_gclear_new:N \g__nm_Hdotsfor_lines_tl
}
```

17.5 The environment {NiceArrayWithDelims}

```
\NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } }
  {
    \str_if_empty:NT \g__nm_type_env_str
      {
        \str_gset:Nn \g__nm_type_env_str
          { environment ~ { NiceArrayWithDelims } }
      }
    \__nm_adapt_S_column:
    \__nm_test_if_math_mode:
    \bool_if:NT \l__nm_in_env_bool { \__nm_fatal:n { Yet~in~env } }
    \bool_set_true:N \l__nm_in_env_bool
```

We deactivate Tikz externalization (since we use Tikz pictures with the options `overlay` and `remember picture`, there would be errors).

```
    \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 extension.

```
    \int_gincr:N \g__nm_env_int
    \bool_if:NF \l__nm_block_auto_columns_width_bool
      { \dim_gzero_new:N \g__nm_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 \__nm_vline: }
```

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` and `b`.

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

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__nm_last_row_int = { -1 }
      {
        \bool_set_true:N \l__nm_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__nm_name_str
          {
            \cs_if_exist:cT { __nm_last_row_ \int_use:N \g__nm_env_int }
              {
                \int_set:Nn \l__nm_last_row_int
                  { \use:c { __nm_last_row_ \int_use:N \g__nm_env_int } }
              }
```

```
          }
          {
            \cs_if_exist:cT { __nm_last_row_ \l__nm_name_str }
              {
                \int_set:Nn \l__nm_last_row_int
                  { \use:c { __nm_last_row_ \l__nm_name_str } }
              }
          }
      }
```

The code in `\@@_pre_array:` is common to `{NiceArrayWithDelims}` and `{NiceMatrix}`.

```
    \__nm_pre_array:
```

We compute the width of the two delimiters.

```
    \dim_gzero_new:N \g__nm_left_delim_dim
    \dim_gzero_new:N \g__nm_right_delim_dim
    \bool_if:NTF \l__nm_NiceArray_bool
      {
        \dim_gset:Nn \g__nm_left_delim_dim { 2 \arraycolsep }
        \dim_gset:Nn \g__nm_right_delim_dim { 2 \arraycolsep }
      }
      {
        \group_begin:
        \dim_set_eq:NN \nulldelimiterspace \c_zero_dim
        \hbox_set:Nn \l_tmpa_box
          {
            \c_math_toggle_token
            \left #1 \vcenter to 3 cm { } \right.
            \c_math_toggle_token

          }
        \dim_gset:Nn \g__nm_left_delim_dim { \box_wd:N \l_tmpa_box }
        \hbox_set:Nn \l_tmpa_box
          {
            \dim_set_eq:NN \nulldelimiterspace \c_zero_dim
            \c_math_toggle_token
            \left. \vcenter to 3 cm { } \right #2
            \c_math_toggle_token
          }
        \dim_gset:Nn \g__nm_right_delim_dim { \box_wd:N \l_tmpa_box }
        \group_end:
      }
```

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__nm_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__nm_first_col_int = \c_zero_int
      { \tl_put_left:NV \l_tmpa_tl \c__nm_preamble_first_col_tl }
      {
        \bool_if:NT \l__nm_NiceArray_bool
          {
            \bool_if:NF \l__nm_exterior_arraycolsep_bool
              { \tl_put_left:Nn \l_tmpa_tl { @ { } } }
          }
      }
    \int_compare:nNnTF \l__nm_last_col_int > { -1 }
      { \tl_put_right:NV \l_tmpa_tl \c__nm_preamble_last_col_tl }
      {
        \bool_if:NT \l__nm_NiceArray_bool
          {
            \bool_if:NF \l__nm_exterior_arraycolsep_bool
              { \tl_put_right:Nn \l_tmpa_tl { @ { } } }
```

```
          }
      }
```

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

```
    \hbox_set:Nw \l__nm_the_array_box
    \skip_horizontal:n \l__nm_left_margin_dim
    \skip_horizontal:n \l__nm_extra_left_margin_dim
    \c_math_toggle_token
```

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 \__nm_array: \l_tmpa_tl
  }
```

We begin the second part of the environment `{NiceArrayWithDelims}`. If all the columns must have the same width (if the user has used the option `columns-width` or the option `auto-column-width` of the environment `{NiceMatrixBlock}`), we add a row in the array to fix the width of the columns and construct the "`col`" nodes `nm-`*a*`-col-`*j* (these nodes will be used by the horizontal open dotted lines and by the commands `\@@_vdottedline:n`).

```
  {
    \bool_if:nT
      {
        \l__nm_auto_columns_width_bool
        || \dim_compare_p:nNn \l__nm_columns_width_dim > \c_zero_dim
      }
      {
        \crcr
        \int_compare:nNnT \l__nm_first_col_int = 0 { \omit & }
        \omit
```

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

```
        \skip_horizontal:N \arraycolsep
        \tikz [ remember~picture , overlay ]
          \coordinate [ name = nm - \int_use:N \g__nm_env_int - col - 0 ] ;
        \skip_horizontal:n { - \arraycolsep }
```

We compute in `\g_tmpa_dim` the common width of the columns. 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_dim`, like all the affectations, must be done after the `\omit` of the cell.

```
        \bool_if:nTF
          {
            \l__nm_auto_columns_width_bool
            && ! \l__nm_block_auto_columns_width_bool
          }
          {
            \dim_gset:Nn \g_tmpa_dim
              { \g__nm_max_cell_width_dim + 2 \arraycolsep }
          }
          {
            \dim_gset:Nn \g_tmpa_dim
              { \l__nm_columns_width_dim + 2 \arraycolsep }
          }
        \skip_horizontal:N \g_tmpa_dim
        \tikz [ remember~picture , overlay ]
          \coordinate [ name = nm - \int_use:N \g__nm_env_int - col - 1 ] ;
```

We begin a loop over the columns. The integer `\g_tmpa_int` will be the number of the current column. This integer is not used to fix the width of the column (since all the columns have the same width equal to `\g_@@_tmpa_dim`) but for the Tikz nodes.

```
        \int_gset:Nn \g_tmpa_int 1
        \bool_if:nTF \g__nm_last_col_found_bool
```

```
          { \prg_replicate:nn { \g__nm_col_total_int - 3 } }
          { \prg_replicate:nn { \g__nm_col_total_int - 2 } }
          {
             &
             \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_dim
```

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

```
             \tikz [ remember~picture , overlay ]
               \coordinate
                 [
                   name = nm - \int_use:N \g__nm_env_int -
                          col - \int_use:N \g_tmpa_int
                 ] ;
          }
```

For the last column, we want a special treatment because of the final `\arraycolsep`.

```
        &
        \omit
        \int_gincr:N \g_tmpa_int
        \skip_horizontal:N \g_tmpa_dim
        \skip_horizontal:n { - \arraycolsep }
        \tikz [ remember~picture , overlay ]
          \coordinate
            [
              name = nm - \int_use:N \g__nm_env_int -
                     col - \int_use:N \g_tmpa_int
            ] ;
        \skip_horizontal:N \arraycolsep
      }
    \endarray
    \c_math_toggle_token
    \skip_horizontal:n \l__nm_right_margin_dim
    \skip_horizontal:n \l__nm_extra_right_margin_dim
    \hbox_set_end:

    \int_compare:nNnT \l__nm_last_row_int > { -2 }
      {
        \bool_if:NF \l__nm_last_row_without_value_bool
          {
            \int_compare:nNnF \l__nm_last_row_int = \c@iRow
              {
                \__nm_error:n { Wrong~last~row }
                \int_gset_eq:NN \l__nm_last_row_int \c@iRow
              }
          }
      }
```

Now, we compute `\l_tmpa_dim` which is the vertical dimension of the "first row" above the array (when the key `first-row` is used).

```
    \int_compare:nNnTF \l__nm_first_row_int = \c_zero_int
      {
        \dim_set:Nn \l_tmpa_dim
          { \g__nm_dp_row_zero_dim + \lineskip + \g__nm_ht_row_zero_dim }
      }
      { \dim_zero:N \l_tmpa_dim }
```

We compute `\l_tmpb_dim` which is the vertical dimension 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".[26]

```
    \int_compare:nNnTF \l__nm_last_row_int > { -2 }
```

[26] A value of -1 for `\l_@@_last_row_int` means that there is a "last row" but the number of that row is unknown (the user have not set the value with the option `last row`).

```
     {
      \dim_set:Nn \l_tmpb_dim
       { \g__nm_ht_last_row_dim + \lineskip + \g__nm_dp_last_row_dim }
     }
     { \dim_zero:N \l_tmpb_dim }
```

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. 45).

```
    \int_compare:nNnT \l__nm_first_col_int = \c_zero_int
     {
      \skip_horizontal:n \arraycolsep
      \skip_horizontal:n \g__nm_width_first_col_dim
     }
```

The construction of the real box is different in `{NiceArray}` and in its variants (`{pNiceArray}`, etc.) because, in `{NiceArray}`, we have to take into account the option of position (`t`, `c` or `b`). We begin with `{NiceArray}`.

```
    \bool_if:NTF \l__nm_NiceArray_bool
     {
      \int_compare:nNnT \l__nm_first_row_int = \c_zero_int
       {
        \str_if_eq:VnTF \l__nm_pos_env_str { t }
         {
          \box_move_up:nn
           { \l_tmpa_dim - \g__nm_ht_row_zero_dim + \g__nm_ht_row_one_dim }
         }
       }
       {
        \int_compare:nNnT \l__nm_last_row_int > 0
         {
          \str_if_eq:VnT \l__nm_pos_env_str { b }
           {
            \box_move_down:nn
             {
              \l_tmpb_dim
              - \g__nm_dp_last_row_dim + \g__nm_dp_ante_last_row_dim
             }
           }
         }
       }
      { \box_use_drop:N \l__nm_the_array_box }
     }
```

Now, in the case of an environment `{pNiceArray}`, `{bNiceArray}`, etc.

```
     {
      \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 size in `\l_tmpa_dim`).

```
          \skip_vertical:n { - \l_tmpa_dim }
          \hbox:n
           {
            \skip_horizontal:n { - \arraycolsep }
            \box_use_drop:N \l__nm_the_array_box
            \skip_horizontal:n { - \arraycolsep }
           }
```

We take into account the "last row" (we have previously computed its size in `\l_tmpb_dim`).

```
          \skip_vertical:n { - \l_tmpb_dim }
         }
```

```
            \right #2
            \c_math_toggle_token
          }
        \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 }
        \box_use_drop:N \l_tmpa_box
      }
```

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. 47).

```
    \bool_if:NT \g__nm_last_col_found_bool
      {
        \skip_horizontal:n \g__nm_width_last_col_dim
        \skip_horizontal:n \arraycolsep
      }
    \__nm_after_array:
  }
```

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

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

```
\tl_const:Nn \c__nm_preamble_first_col_tl
  {
    >
      {
        \__nm_begin_of_row:
```

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_tmpa_box
        \c_math_toggle_token
        \bool_if:NT \l__nm_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_if:nT
          {
            \int_compare_p:nNn \c@iRow > \c_zero_int
            &&
            (
              \int_compare_p:nNn \l__nm_last_row_int < 0
              ||
              \int_compare_p:nNn \c@iRow < \l__nm_last_row_int
            )
          }
          { \l__nm_code_for_first_col_tl }
      }
    l
    <
      {
        \c_math_toggle_token
        \hbox_set_end:
        \__nm_actualization_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__nm_width_first_col_dim
          {
            \dim_max:nn
              \g__nm_width_first_col_dim
              { \box_wd:N \l_tmpa_box }
          }
```

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

```
        \hbox_overlap_left:n
          {
```

```
            \tikz
              [
                remember~picture ,
                inner~sep = \c_zero_dim ,
                minimum~width = \c_zero_dim ,
                baseline
              ]
            \node
              [
                anchor = base ,
                name =
                  nm -
                  \int_use:N \g__nm_env_int -
                  \int_use:N \c@iRow -
                  0 ,
                alias =
                  \str_if_empty:NF \l__nm_name_str
                    {
                      \l__nm_name_str -
                      \int_use:N \c@iRow -
                      0
                    }
              ]
              { \box_use:N \l_tmpa_box } ;
            \skip_horizontal:n
              {
                \g__nm_left_delim_dim +
                \l__nm_left_margin_dim +
                \l__nm_extra_left_margin_dim
              }
          }
        \skip_horizontal:n { - 2 \arraycolsep }
      }
  }
```

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

```
\tl_const:Nn \c__nm_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__nm_last_col_found_bool
        \int_gincr:N \c@jCol
        \int_gset:Nn \g__nm_col_total_int
          { \int_max:nn \g__nm_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_tmpa_box
          \c_math_toggle_token
          \bool_if:NT \l__nm_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”.

```
        \bool_if:nT
          {
            \int_compare_p:nNn \c@iRow > \c_zero_int
            &&
            (
              \int_compare_p:nNn \l__nm_last_row_int < 0
              ||
              \int_compare_p:nNn \c@iRow < \l__nm_last_row_int
            )
          }
          { \l__nm_code_for_last_col_tl }
```

```
      }
    l
    <
      {
        \c_math_toggle_token
        \hbox_set_end:
        \__nm_actualization_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__nm_width_last_col_dim
           {
             \dim_max:nn
               \g__nm_width_last_col_dim
               { \box_wd:N \l_tmpa_box }
           }
        \skip_horizontal:n { - 2 \arraycolsep }
```

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

```
        \hbox_overlap_right:n
          {
            \skip_horizontal:n
              {
                \g__nm_right_delim_dim +
                \l__nm_right_margin_dim +
                \l__nm_extra_right_margin_dim
              }
            \tikz
              [
                remember~picture ,
                inner~sep = \c_zero_dim ,
                minimum~width = \c_zero_dim ,
                baseline
              ]
            \node
              [
                anchor = base ,
                name =
                  nm -
                  \int_use:N \g__nm_env_int -
                  \int_use:N \c@iRow -
                  \int_use:N \c@jCol ,
                alias =
                  \str_if_empty:NF \l__nm_name_str
                    {
                      \l__nm_name_str -
                      \int_use:N \c@iRow -
                      \int_use:N \c@jCol
                    }
              ]
              { \box_use:N \l_tmpa_box } ;
          }
      }
  }
```

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__nm_NiceArray_bool
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { 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}. These variants exist since the version 3.0 of nicematrix.

```
\NewDocumentEnvironment { pNiceArray } { }
  {
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { pNiceArray } } }
    \__nm_test_if_math_mode:
    \NiceArrayWithDelims ( )
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { bNiceArray } { }
  {
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { bNiceArray } } }
    \__nm_test_if_math_mode:
    \NiceArrayWithDelims [ ]
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { BNiceArray } { }
  {
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { BNiceArray } } }
    \__nm_test_if_math_mode:
    \NiceArrayWithDelims \{ \}
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { vNiceArray } { }
  {
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { vNiceArray } } }
    \__nm_test_if_math_mode:
    \NiceArrayWithDelims | |
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { VNiceArray } { }
  {
    \str_if_empty:NT \g__nm_type_env_str
      { \str_gset:Nn \g__nm_type_env_str { environment ~ { VNiceArray } } }
    \__nm_test_if_math_mode:
    \NiceArrayWithDelims \| \|
  }
  { \endNiceArrayWithDelims }
```

17.6 The environment {NiceMatrix} and its variants

```
\cs_new_protected:Npn \__nm_define_env:n #1
  {
    \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
      {
        \str_gset:Nn \g__nm_type_env_str { environment ~ { #1 NiceMatrix } }
        \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
        \begin { #1 NiceArray }
          {
            *
```

```
       {
        \int_compare:nNnTF \l__nm_last_col_int = { -1 }
         \c@MaxMatrixCols
         { \int_eval:n { \l__nm_last_col_int - 1 } }
       }
       C
     }
   }
   { \end { #1 NiceArray } }
 }
```

```
\__nm_define_env:n { }
\__nm_define_env:n p
\__nm_define_env:n b
\__nm_define_env:n B
\__nm_define_env:n v
\__nm_define_env:n V
```

17.7 How to know whether a cell is "empty"

The conditionnal `\@@_if_not_empty_cell:nnT` tests whether a cell is empty. The first two arguments must be LaTeX3 counters for the row and the column of the considered cell.

```
\prg_set_conditional:Npnn \__nm_if_not_empty_cell:nn #1 #2 { T , TF }
 {
```

First, we want to test whether the cell is in the virtual sequence of "non-empty" cells. There are several important remarks:

- we don't use a `expl3` sequence for efficiency;
- the "non-empty" cells in this sequence are not, in fact, all the non-empty cells of the array: on the contrary they are only cells declared as non-empty for a special reason (as of now, there are only cells which are on a dotted line which is already drawn or which will be drawn "just after");
- the flag `\l_tmpa_bool` will be raised when the cell is actually on this virtual sequence.

```
  \bool_set_false:N \l_tmpa_bool
  \cs_if_exist:cTF
   { __nm _ dotted _ \int_use:N #1 - \int_use:N #2 }
   \prg_return_true:
   {
```

We know that the cell is not in the virtual sequence of the "non-empty" cells. Now, we test wether the cell is a "virtual cell", that is to say a cell after the `\\` of the line of the array. It's easy to known whether a cell is vitual: the cell is virtual if, and only if, the corresponding Tikz node doesn't exist.

```
    \cs_if_free:cTF
     {
      pgf@sh@ns@nm -
      \int_use:N \g__nm_env_int -
      \int_use:N #1 -
      \int_use:N #2
     }
     { \prg_return_false: }
     {
```

Now, we want to test whether the cell is in the virtual sequence of "empty" cells. There are several important remarks:

- we don't use a `expl3` sequence for efficiency ;
- the "empty" cells in this sequence are not, in fact, all the non-empty cells of the array: on the contrary they are only cells declared as non-empty for a special reason ;
- the flag `\l_tmpa_bool` will be raised when the cell is actually on this virtual sequence.

```
            \bool_set_false:N \l_tmpa_bool
            \cs_if_exist:cT
              { __nm _ empty _ \int_use:N #1 - \int_use:N #2 }
              {
                \int_compare:nNnT
                  { \use:c { __nm _ empty _ \int_use:N #1 - \int_use:N #2 } }
                  =
                  \g__nm_env_int
                { \bool_set_true:N \l_tmpa_bool }
              }
            \bool_if:NTF \l_tmpa_bool
              \prg_return_false:
```

In the general case, we consider the width of the Tikz node corresponding to the cell. In order to compute this width, we have to extract the coordinate of the west and east anchors of the node. This extraction needs a command environment `{pgfpicture}` but, in fact, nothing is drawn.

```
              {
                \begin { pgfpicture }
```

We store the name of the node corresponding to the cell in `\l_tmpa_tl`.

```
                \tl_set:Nx \l_tmpa_tl
                   {
                     nm -
                     \int_use:N \g__nm_env_int -
                     \int_use:N #1 -
                     \int_use:N #2
                   }
                \pgfpointanchor \l_tmpa_tl { east }
                \dim_gset:Nn \g_tmpa_dim \pgf@x
                \pgfpointanchor \l_tmpa_tl { west }
                \dim_gset:Nn \g_tmpb_dim \pgf@x
                \end { pgfpicture }
                \dim_compare:nNnTF
                  { \dim_abs:n { \g_tmpb_dim - \g_tmpa_dim } } < { 0.5 pt }
                  \prg_return_false:
                  \prg_return_true:
              }
          }
      }
  }
```

17.8 After the construction of the array

```
\cs_new_protected:Nn \__nm_after_array:
  {
    \int_compare:nNnTF \c@iRow > \c_zero_int
      \__nm_after_array_i:
      {
        \__nm_error:n { Zero~row }
        \__nm_restore_iRow_jCol:
      }
  }
```

We deactivate Tikz externalization (since we use Tikz pictures with the options `overlay` and `remember picture`, there would be errors).

```
\cs_new_protected:Nn \__nm_after_array_i:
  {
    \group_begin:
    \cs_if_exist:NT \tikz@library@external@loaded
      { \tikzset { external / export = false } }
```

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".[27]

```
    \int_gset_eq:NN \c@jCol \g__nm_col_total_int
    \bool_if:nT \g__nm_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__nm_row_total_int \c@iRow
    \int_compare:nNnT \l__nm_last_row_int > { -1 }
      { \int_gsub:Nn \c@iRow \c_one_int }
```

In the user has 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__nm_last_row_without_value_bool
      {
        \iow_now:Nn \@mainaux \ExplSyntaxOn
        \iow_now:Nx \@mainaux
          {
            \cs_gset:cpn { __nm_last_row_ \int_use:N \g__nm_env_int }
              { \int_use:N \g__nm_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__nm_name_str
          {
            \iow_now:Nx \@mainaux
              {
                \cs_gset:cpn { __nm_last_row_ \l__nm_name_str }
                  { \int_use:N \g__nm_row_total_int }
              }
          }
        \iow_now:Nn \@mainaux \ExplSyntaxOff
      }
```

By default, the diagonal lines will be parallelized[28]. 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__nm_parallelize_diags_bool
      {
        \int_zero_new:N \l__nm_ddots_int
        \int_zero_new:N \l__nm_iddots_int
```

The dimensions \l_@@_delta_x_one_dim and \l_@@_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 \l_@@_delta_x_two_dim and \l_@@_delta_y_two_dim are the Δ_x and Δ_y of the first \Iddots diagonal.

```
        \dim_zero_new:N \l__nm_delta_x_one_dim
        \dim_zero_new:N \l__nm_delta_y_one_dim
        \dim_zero_new:N \l__nm_delta_x_two_dim
        \dim_zero_new:N \l__nm_delta_y_two_dim
      }
```

If the user has used the option create-extra-nodes, the "medium nodes" and "large nodes" are created. We recall that the command \@@_create_extra_nodes:, when used once, becomes no-op (in the current TeX group).

```
    \bool_if:NT \g__nm_extra_nodes_bool \__nm_create_extra_nodes:
    \int_zero_new:N \l__nm_initial_i_int
    \int_zero_new:N \l__nm_initial_j_int
    \int_zero_new:N \l__nm_final_i_int
    \int_zero_new:N \l__nm_final_j_int
    \bool_set_false:N \l__nm_initial_open_bool
    \bool_set_false:N \l__nm_final_open_bool
```

[27]We remind that the potential "first column" has the number 0.
[28]It's possible to use the option parallelize-diags to disable this parallelization.

If the option `small` is used, the values `\l_@@_radius_dim` and `\l_@@_inter_dots_dim` (used to draw the dotted lines) are changed.

```
    \bool_if:NT \l__nm_small_bool
      {
        \dim_set:Nn \l__nm_radius_dim { 0.37 pt }
        \dim_set:Nn \l__nm_inter_dots_dim { 0.25 em }
      }
```

Now, we really draw the lines. The code to draw the lines has been constructed in the token lists `\g_@@_Vdots_lines_tl`, etc.

```
    \g__nm_Hdotsfor_lines_tl
    \g__nm_Vdots_lines_tl
    \g__nm_Ddots_lines_tl
    \g__nm_Iddots_lines_tl
    \g__nm_Cdots_lines_tl
    \g__nm_Ldots_lines_tl
```

Now, the `code-after`.

```
    \tikzset
      {
        every~picture / .style =
          {
            overlay ,
            remember~picture ,
            name~prefix = nm - \int_use:N \g__nm_env_int -
          }
      }
    \cs_set_eq:NN \line \__nm_line:nn
    \g__nm_code_after_tl
    \tl_gclear:N \g__nm_code_after_tl
    \group_end:
    \str_gclear:N \g__nm_type_env_str
    \__nm_restore_iRow_jCol:
  }
\cs_new_protected:Nn \__nm_restore_iRow_jCol:
  {
    \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l__nm_save_iRow_int }
    \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l__nm_save_jCol_int }
  }
```

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.

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

For a closed extremity, we use the normal node and for a open one, we use the "medium node" or, if it exists, the `w` node (the medium and large nodes are created with `\@@_create_extra_nodes:` if they have not been created yet).

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x-value of the orientation vector of the line;

- the fourth argument is the y-value of the orientation vector of the line;

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
\cs_new_protected:Nn \__nm_find_extremities_of_line:nnnn
  {
```

First, we declare the current cell as "dotted" because we forbide intersections of dotted lines.

```
    \cs_set:cpn { __nm _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
    \int_set:Nn \l__nm_initial_i_int { #1 }
    \int_set:Nn \l__nm_initial_j_int { #2 }
    \int_set:Nn \l__nm_final_i_int { #1 }
    \int_set:Nn \l__nm_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.

```
    \bool_set_false:N \l__nm_stop_loop_bool
    \bool_do_until:Nn \l__nm_stop_loop_bool
      {
        \int_add:Nn \l__nm_final_i_int { #3 }
        \int_add:Nn \l__nm_final_j_int { #4 }
```

We test if we are still in the matrix.

```
        \bool_set_false:N \l__nm_final_open_bool
        \int_compare:nNnTF \l__nm_final_i_int > \c@iRow
          {
            \int_compare:nNnT { #3 } = 1
              { \bool_set_true:N \l__nm_final_open_bool }
          }
          {
            \int_compare:nNnTF \l__nm_final_j_int < 1
              {
                \int_compare:nNnT { #4 } = { -1 }
                  { \bool_set_true:N \l__nm_final_open_bool }
              }
              {
                \int_compare:nNnT \l__nm_final_j_int > \c@jCol
                  {
                    \int_compare:nNnT { #4 } = 1
                      { \bool_set_true:N \l__nm_final_open_bool }
                  }
              }
          }
        \bool_if:NTF \l__nm_final_open_bool
```

If we are outside the matrix, we have found the extremity of the dotted line and it's a *open* extremity.

```
          {
```

We do a step backwards because we will draw the dotted line upon the last cell in the matrix (we will use the "medium node" of this cell).

```
            \int_sub:Nn \l__nm_final_i_int { #3 }
            \int_sub:Nn \l__nm_final_j_int { #4 }
            \bool_set_true:N \l__nm_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`.

```
          {
            \__nm_if_not_empty_cell:nnTF \l__nm_final_i_int \l__nm_final_j_int
              { \bool_set_true:N \l__nm_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.

```
              {
                \cs_set:cpn
                  {
                    __nm _ dotted _
                    \int_use:N \l__nm_final_i_int -
                    \int_use:N \l__nm_final_j_int
                  }
                  { }
              }
          }
      }
```

We test wether the initial extremity of the dotted line is an implicit cell already dotted (by another dotted line). In this case, we can't draw the line because we have no Tikz node at the extremity of the arrow (and we can't use the "medium node" or the "large node" because we should use the normal node since the extremity is not open).

```
    \cs_if_free:cT
      {
        pgf@sh@ns@nm -
        \int_use:N \g__nm_env_int -
        \int_use:N \l__nm_final_i_int -
        \int_use:N \l__nm_final_j_int
      }
      {
        \bool_if:NF \l__nm_final_open_bool
          {
            \msg_error:nnx { nicematrix } { Impossible~line }
              { \int_use:N \l__nm_final_i_int }
            \bool_set_true:N \l__nm_impossible_line_bool
          }
      }
```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programmation is similar to the previous one.

```
    \bool_set_false:N \l__nm_stop_loop_bool
    \bool_do_until:Nn \l__nm_stop_loop_bool
      {
        \int_sub:Nn \l__nm_initial_i_int { #3 }
        \int_sub:Nn \l__nm_initial_j_int { #4 }
        \bool_set_false:N \l__nm_initial_open_bool
        \int_compare:nNnTF \l__nm_initial_i_int < 1
          {
            \int_compare:nNnT { #3 } = 1
              { \bool_set_true:N \l__nm_initial_open_bool }
          }
          {
            \int_compare:nNnTF \l__nm_initial_j_int < 1
              {
                \int_compare:nNnT { #4 } = 1
                  { \bool_set_true:N \l__nm_initial_open_bool }
              }
              {
                \int_compare:nNnT \l__nm_initial_j_int > \c@jCol
```

```
                {
                  \int_compare:nNnT { #4 } = { -1 }
                    { \bool_set_true:N \l__nm_initial_open_bool }
                }
            }
        }
      \bool_if:NTF \l__nm_initial_open_bool
        {
          \int_add:Nn \l__nm_initial_i_int { #3 }
          \int_add:Nn \l__nm_initial_j_int { #4 }
          \bool_set_true:N \l__nm_stop_loop_bool
        }
        {
          \__nm_if_not_empty_cell:nnTF
            \l__nm_initial_i_int \l__nm_initial_j_int
            { \bool_set_true:N \l__nm_stop_loop_bool }
            {
              \cs_set:cpn
                {
                  __nm _ dotted _
                  \int_use:N \l__nm_initial_i_int -
                  \int_use:N \l__nm_initial_j_int
                }
                { }
            }
        }
    }
```

We test wether the initial extremity of the dotted line is an implicit cell already dotted (by another dotted line). In this case, we can't draw the line because we have no Tikz node at the extremity of the arrow (and we can't use the "medium node" or the "large node" because we should use the normal node since the extremity is not open).

```
  \cs_if_free:cT
    {
      pgf@sh@ns@nm -
      \int_use:N \g__nm_env_int -
      \int_use:N \l__nm_initial_i_int -
      \int_use:N \l__nm_initial_j_int
    }
    {
      \bool_if:NF \l__nm_initial_open_bool
        {
          \msg_error:nnx { nicematrix } { Impossible~line }
            { \int_use:N \l__nm_initial_i_int }
          \bool_set_true:N \l__nm_impossible_line_bool
        }
    }
```

If we have at least one open extremity, we create the "medium nodes" in the matrix[29]. We remind that, when used once, the command `\@@_create_extra_nodes:` becomes no-op in the current TeX group.

```
  \bool_if:nT \l__nm_initial_open_bool \__nm_create_extra_nodes:
  \bool_if:NT \l__nm_final_open_bool \__nm_create_extra_nodes:
}
```

The command `\@@_retrieve_coords:nn` retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw [30]. This command has four implicit arguments which are `\l_@@_initial_i_int`, `\l_@@_initial_j_int`, `\l_@@_final_i_int` and `\l_@@_final_j_int`.

[29] We should change this. Indeed, for an open extremity of an *horizontal* dotted line, we use the `w` node, if, it exists, and not the "medium node".

[30] In fact, with diagonal lines, or vertical lines in columns of type `L` or `R`, an adjustment of one of the coordinates may be done.

The two arguments of the command `\@@_retrieve_coords:nn` are the suffix and the anchor that must be used for the two nodes.
The coordinates are stored in `\g_@@_x_initial_dim`, `\g_@@_y_initial_dim`, `\g_@@_x_final_dim`, `\g_@@_y_final_dim`. These variables are global for technical reasons: we have to do an affectation in an environment `{tikzpicture}`.

```
\cs_new_protected:Nn \__nm_retrieve_coords:nn
  {
    \dim_gzero_new:N \g__nm_x_initial_dim
    \dim_gzero_new:N \g__nm_y_initial_dim
    \dim_gzero_new:N \g__nm_x_final_dim
    \dim_gzero_new:N \g__nm_y_final_dim
    \begin { tikzpicture } [ remember~picture ]
      \tikz@parse@node \pgfutil@firstofone
        ( nm - \int_use:N \g__nm_env_int -
               \int_use:N \l__nm_initial_i_int -
               \int_use:N \l__nm_initial_j_int #1 )
      \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
      \dim_gset:Nn \g__nm_y_initial_dim \pgf@y
      \tikz@parse@node \pgfutil@firstofone
        ( nm - \int_use:N \g__nm_env_int -
               \int_use:N \l__nm_final_i_int -
               \int_use:N \l__nm_final_j_int #2 )
      \dim_gset:Nn \g__nm_x_final_dim \pgf@x
      \dim_gset:Nn \g__nm_y_final_dim \pgf@y
    \end { tikzpicture }
  }
\cs_generate_variant:Nn \__nm_retrieve_coords:nn { x x }
```

For the horizontal lines with open extremities, we must take into account the "`col`" nodes created in the environments which have a fixed width of the columns. The following command will recompute the x-value of the extremities in this case (erasing the value computed in `\@@_retrieve_coords:nn`).

```
\cs_new_protected:Nn \__nm_adjust_with_col_nodes:
  {
    \bool_if:NT \l__nm_initial_open_bool
      {
        \cs_if_exist:cT
          { pgf@sh@ns@nm - \int_use:N \g__nm_env_int - col - 0 }
          {
            \begin { tikzpicture } [ remember~picture ]
            \tikz@parse@node \pgfutil@firstofone
              ( nm - \int_use:N \g__nm_env_int - col - 0 )
            \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
            \end { tikzpicture }
          }
      }
    \bool_if:NT \l__nm_final_open_bool
      {
        \cs_if_exist:cT
          {
            pgf@sh@ns@nm - \int_use:N \g__nm_env_int - col -
            \int_use:N \c@jCol
          }
          {
            \begin { tikzpicture } [ remember~picture ]
            \tikz@parse@node \pgfutil@firstofone
              (
                nm - \int_use:N \g__nm_env_int - col -
                \int_use:N \c@jCol
              )
            \dim_gset:Nn \g__nm_x_final_dim \pgf@x
            \end { tikzpicture }
          }
```

```
    }
  }
```

```
\cs_new_protected:Nn \__nm_draw_Ldots:nn
  {
    \cs_if_free:cT { __nm _ dotted _ #1 - #2 }
      {
        \bool_set_false:N \l__nm_impossible_line_bool
        \__nm_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
        \bool_if:NF \l__nm_impossible_line_bool \__nm_actually_draw_Ldots:
      }
  }
```

The command `\@@_actually_draw_Ldots:` draws the `Ldots` line using `\l_@@_initial_i_int`, `\l_@@_initial_j_int`, `\l_@@_initial_open_bool`, `\l_@@_final_i_int`, `\l_@@_final_j_int` and `\l_@@_final_open_bool`. We have a dedicated command because if is used also by `\Hdotsfor`.

```
\cs_new_protected:Nn \__nm_actually_draw_Ldots:
  {
    \__nm_retrieve_coords:xx
      {
        \bool_if:NTF \l__nm_initial_open_bool
          {
```

If a `w` node exists we use the `w` node for the extremity.

```
           \cs_if_exist:cTF
             {
               pgf@sh@ns@nm
               - \int_use:N \g__nm_env_int
               - \int_use:N \l__nm_initial_i_int
               - \int_use:N \l__nm_initial_j_int - w
             }
             { - w.base~west }
             { - medium.base~west }
         }
         { .base~east }
      }
      {
        \bool_if:NTF \l__nm_final_open_bool
          {
            \cs_if_exist:cTF
              {
                pgf@sh@ns@nm
                - \int_use:N \g__nm_env_int
                - \int_use:N \l__nm_final_i_int
                - \int_use:N \l__nm_final_j_int - w
              }
              { - w.base~east }
              { - medium.base~east }
          }
          { .base~west }
      }
    \__nm_adjust_with_col_nodes:
    \bool_if:NT \l__nm_initial_open_bool
      { \dim_gset_eq:NN \g__nm_y_initial_dim \g__nm_y_final_dim }
    \bool_if:NT \l__nm_final_open_bool
      { \dim_gset_eq:NN \g__nm_y_final_dim \g__nm_y_initial_dim }
```

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.

```
    \dim_gadd:Nn \g__nm_y_initial_dim { 0.53 pt }
    \dim_gadd:Nn \g__nm_y_final_dim { 0.53 pt }
    \__nm_draw_tikz_line:
  }
```

```
\cs_new_protected:Nn \__nm_draw_Cdots:nn
  {
    \cs_if_free:cT { __nm _ dotted _ #1 - #2 }
      {
        \bool_set_false:N \l__nm_impossible_line_bool
        \__nm_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
        \bool_if:NF \l__nm_impossible_line_bool
          {
            \__nm_retrieve_coords:xx
              {
                \bool_if:NTF \l__nm_initial_open_bool
                  {
                    \cs_if_exist:cTF
                      {
                        pgf@sh@ns@nm
                        - \int_use:N \g__nm_env_int
                        - \int_use:N \l__nm_initial_i_int
                        - \int_use:N \l__nm_initial_j_int - w
                      }
                      { - w.mid~west }
                      { - medium.mid~west }
                  }
                  { .mid~east }
              }
              {
                \bool_if:NTF \l__nm_final_open_bool
                  {
                    \cs_if_exist:cTF
                      {
                        pgf@sh@ns@nm
                        - \int_use:N \g__nm_env_int
                        - \int_use:N \l__nm_final_i_int
                        - \int_use:N \l__nm_final_j_int - w
                      }
                      { - w.mid~east }
                      { - medium.mid~east }
                  }
                  { .mid~west }
              }
            \__nm_adjust_with_col_nodes:
            \bool_if:NT \l__nm_initial_open_bool
              { \dim_gset_eq:NN \g__nm_y_initial_dim \g__nm_y_final_dim }
            \bool_if:NT \l__nm_final_open_bool
              { \dim_gset_eq:NN \g__nm_y_final_dim \g__nm_y_initial_dim }
            \__nm_draw_tikz_line:
          }
      }
  }
```

For the vertical dots, we have to distinguish different instances because we want really vertical lines. Be careful: it's not possible to insert the command `\@@_retrieve_coords:nn` in the arguments `T` and `F` of the expl3 commands (why?).

```
\cs_new_protected:Nn \__nm_draw_Vdots:nn
  {
    \cs_if_free:cT { __nm _ dotted _ #1 - #2 }
      {
        \bool_set_false:N \l__nm_impossible_line_bool
        \__nm_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_zero_int
        \bool_if:NF \l__nm_impossible_line_bool
          {
            \__nm_retrieve_coords:xx
              {
                \bool_if:NTF \l__nm_initial_open_bool
```

```
                  { - medium.north~west }
                  { .south~west }
              }
              {
                \bool_if:NTF \l__nm_final_open_bool
                  { - medium.south~west }
                  { .north~west }
              }
```

The boolean `\l_tmpa_bool` indicates whether the column is of type `l` (`L` of `{NiceArray}`) or may be considered as if.

```
            \bool_set:Nn \l_tmpa_bool
              { \dim_compare_p:nNn \g__nm_x_initial_dim = \g__nm_x_final_dim }
            \__nm_retrieve_coords:xx
              {
                \bool_if:NTF \l__nm_initial_open_bool
                  { - medium.north }
                  { .south }
              }
              {
                \bool_if:NTF \l__nm_final_open_bool
                  { - medium.south }
                  { .north }
              }
```

The boolean `\l_tmpb_bool` indicates whether the column is of type `c` (`C` of `{NiceArray}`) or may be considered as if.

```
            \bool_set:Nn \l_tmpb_bool
              { \dim_compare_p:nNn \g__nm_x_initial_dim = \g__nm_x_final_dim }
            \bool_if:NF \l_tmpb_bool
              {
                \dim_gset:Nn \g__nm_x_initial_dim
                  {
                    \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
                      \g__nm_x_initial_dim \g__nm_x_final_dim
                  }
                \dim_gset_eq:NN \g__nm_x_final_dim \g__nm_x_initial_dim
              }
            \__nm_draw_tikz_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.

```
\cs_new_protected:Nn \__nm_draw_Ddots:nn
  {
    \cs_if_free:cT { __nm _ dotted _ #1 - #2 }
      {
        \bool_set_false:N \l__nm_impossible_line_bool
        \__nm_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_one_int
        \bool_if:NF \l__nm_impossible_line_bool
          {
            \__nm_retrieve_coords:xx
              {
                \bool_if:NTF \l__nm_initial_open_bool
                  { - medium.north~west }
                  { .south~east }
              }
              {
                \bool_if:NTF \l__nm_final_open_bool
                  { - medium.south~east }
```

```
                  { .north~west }
              }
```

We have retrieved the coordinates in the usual way (they are stored in `\g_@@_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__nm_parallelize_diags_bool
              {
                \int_incr:N \l__nm_ddots_int
```

We test if the diagonal line is the first one (the counter `\l_@@_ddots_int` is created for this usage).

```
                \int_compare:nNnTF \l__nm_ddots_int = \c_one_int
```

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_set:Nn \l__nm_delta_x_one_dim
                      { \g__nm_x_final_dim - \g__nm_x_initial_dim }
                    \dim_set:Nn \l__nm_delta_y_one_dim
                      { \g__nm_y_final_dim - \g__nm_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 `\g_@@_y_initial_dim`.

```
                  {
                    \dim_gset:Nn \g__nm_y_final_dim
                      {
                        \g__nm_y_initial_dim +
                        ( \g__nm_x_final_dim - \g__nm_x_initial_dim ) *
                        \dim_ratio:nn \l__nm_delta_y_one_dim \l__nm_delta_x_one_dim
                      }
                  }
              }
```

Now, we can draw the dotted line (after a possible change of `\g_@@_y_initial_dim`).

```
            \__nm_draw_tikz_line:
          }
      }
  }
```

We draw the `\Iddots` diagonals in the same way.

```
\cs_new_protected:Nn \__nm_draw_Iddots:nn
  {
    \cs_if_free:cT { __nm _ dotted _ #1 - #2 }
      {
        \bool_set_false:N \l__nm_impossible_line_bool
        \__nm_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
        \bool_if:NF \l__nm_impossible_line_bool
          {
            \__nm_retrieve_coords:xx
              {
                \bool_if:NTF \l__nm_initial_open_bool
                  { - medium.north~east }
                  { .south~west }
              }
              {
                \bool_if:NTF \l__nm_final_open_bool
                  { - medium.south~west }
                  { .north~east }
              }
            \bool_if:NT \l__nm_parallelize_diags_bool
              {
                \int_incr:N \l__nm_iddots_int
                \int_compare:nNnTF \l__nm_iddots_int = \c_one_int
```

```
                {
                  \dim_set:Nn \l__nm_delta_x_two_dim
                    { \g__nm_x_final_dim - \g__nm_x_initial_dim }
                  \dim_set:Nn \l__nm_delta_y_two_dim
                    { \g__nm_y_final_dim - \g__nm_y_initial_dim }
                }
                {
                  \dim_gset:Nn \g__nm_y_final_dim
                     {
                       \g__nm_y_initial_dim +
                       ( \g__nm_x_final_dim - \g__nm_x_initial_dim ) *
                       \dim_ratio:nn \l__nm_delta_y_two_dim \l__nm_delta_x_two_dim
                     }
                }
            }
          \__nm_draw_tikz_line:
        }
    }
}
```

17.9 The actual instructions for drawing the dotted line with Tikz

The command `\@@_draw_tikz_line:` draws the line using four implicit arguments: `\g_@@_x_initial_dim`, `\g_@@_y_initial_dim`, `\g_@@_x_final_dim` and `\g_@@_y_final_dim`. These variables are global for technical reasons: their first affectation was in an instruction `\tikz`.

```
\cs_new_protected:Nn \__nm_draw_tikz_line:
  {
```

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__nm_l_dim
    \dim_set:Nn \l__nm_l_dim
      {
        \fp_to_dim:n
          {
            sqrt
             (
               (   \dim_use:N \g__nm_x_final_dim
                 - \dim_use:N \g__nm_x_initial_dim
               ) ^ 2
                  +
               (   \dim_use:N \g__nm_y_final_dim
                 - \dim_use:N \g__nm_y_initial_dim
               ) ^ 2
             )
          }
      }
```

We draw only if the length is not equal to zero (in fact, in the first compilation, the length may be equal to zero).

```
    \dim_compare:nNnF \l__nm_l_dim = \c_zero_dim
```

The integer `\l_tmpa_int` is the number of dots of the dotted line.

```
      {
        \bool_if:NTF \l__nm_initial_open_bool
          {
            \bool_if:NTF \l__nm_final_open_bool
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn \l__nm_l_dim \l__nm_inter_dots_dim }
              }
              {
```

```
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l__nm_l_dim - 0.3 em } \l__nm_inter_dots_dim }
              }
          }
          {
            \bool_if:NTF \l__nm_final_open_bool
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l__nm_l_dim - 0.3 em } \l__nm_inter_dots_dim }
              }
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l__nm_l_dim - 0.6 em } \l__nm_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
          {
            ( \g__nm_x_final_dim - \g__nm_x_initial_dim ) *
            \dim_ratio:nn \l__nm_inter_dots_dim \l__nm_l_dim
          }
        \dim_set:Nn \l_tmpb_dim
          {
            ( \g__nm_y_final_dim - \g__nm_y_initial_dim ) *
            \dim_ratio:nn \l__nm_inter_dots_dim \l__nm_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__nm_initial_open_bool
              { \bool_if:NTF \l__nm_final_open_bool 1 0 }
              { \bool_if:NTF \l__nm_final_open_bool 2 1 }
          }
```

In the loop over the dots (`\int_step_inline:nnnn`), the dimensions `\g_@@_x_initial_dim` and `\g_@@_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 \g__nm_x_initial_dim
          {
            ( \g__nm_x_final_dim - \g__nm_x_initial_dim ) *
            \dim_ratio:nn
            { \l__nm_l_dim - \l__nm_inter_dots_dim * \l_tmpa_int }
            { \l__nm_l_dim * 2 }
            * \l_tmpb_int
          }
```

(In a multiplication of a dimension and an integer, the integer must always be put in second position.)

```
        \dim_gadd:Nn \g__nm_y_initial_dim
          {
            ( \g__nm_y_final_dim - \g__nm_y_initial_dim ) *
            \dim_ratio:nn
              { \l__nm_l_dim - \l__nm_inter_dots_dim * \l_tmpa_int }
              { \l__nm_l_dim * 2 } *
            \l_tmpb_int
          }
        \begin { tikzpicture } [ overlay ]
          \int_step_inline:nnn 0 \l_tmpa_int
            {
```

```
            \pgfpathcircle
              { \pgfpoint { \g__nm_x_initial_dim } { \g__nm_y_initial_dim } }
              { \l__nm_radius_dim }
            \pgfusepath { fill }
            \dim_gadd:Nn \g__nm_x_initial_dim \l_tmpa_dim
            \dim_gadd:Nn \g__nm_y_initial_dim \l_tmpb_dim
          }
      \end { tikzpicture }
    }
  }
```

17.10 User commands available in the new environments

We give new names for the commands `\ldots`, `\cdots`, `\vdots` and `\ddots` because these commands will be redefined (if the option `renew-dots` is used).

```
\cs_set_eq:NN \__nm_ldots \ldots
\cs_set_eq:NN \__nm_cdots \cdots
\cs_set_eq:NN \__nm_vdots \vdots
\cs_set_eq:NN \__nm_ddots \ddots
\cs_set_eq:NN \__nm_iddots \iddots
```

The command `\@@_add_to_empty_cells:` adds the current cell to `\g_@@_empty_cells_seq` which is the list of the empty cells (the cells explicitly declared "empty": there may be, of course, other empty cells in the matrix).

```
\cs_new_protected:Nn \__nm_add_to_empty_cells:
  {
    \cs_gset:cpx
      { __nm _ empty _ \int_use:N \c@iRow - \int_use:N \c@jCol }
      { \int_use:N \g__nm_env_int }
  }
```

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 they are still available.

```
\NewDocumentCommand \__nm_Ldots { s }
  {
    \bool_if:nF { #1 } { \__nm_instruction_of_type:n { Ldots } }
    \bool_if:NF \l__nm_nullify_dots_bool { \phantom \__nm_ldots }
    \__nm_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \__nm_Cdots { s }
  {
    \bool_if:nF { #1 } { \__nm_instruction_of_type:n { Cdots } }
    \bool_if:NF \l__nm_nullify_dots_bool { \phantom \__nm_cdots }
    \__nm_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \__nm_Vdots { s }
  {
    \bool_if:nF { #1 } { \__nm_instruction_of_type:n { Vdots } }
    \bool_if:NF \l__nm_nullify_dots_bool { \phantom \__nm_vdots }
    \__nm_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \__nm_Ddots { s }
  {
    \bool_if:nF { #1 } { \__nm_instruction_of_type:n { Ddots } }
    \bool_if:NF \l__nm_nullify_dots_bool { \phantom \__nm_ddots }
    \__nm_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \__nm_Iddots { s }
  {
    \bool_if:nF { #1 } { \__nm_instruction_of_type:n { Iddots } }
    \bool_if:NF \l__nm_nullify_dots_bool { \phantom \__nm_iddots }
    \__nm_add_to_empty_cells:
  }
```

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

```
\cs_new_protected:Nn \__nm_Hspace:
  {
   \__nm_add_to_empty_cells:
   \hspace
  }
```

In the environment `{NiceArray}`, the command `\multicolumn` will be linked to the following command `\@@_multicolumn:nnn`.

```
\cs_set_eq:NN \__nm_old_multicolumn \multicolumn
\cs_new:Npn \__nm_multicolumn:nnn #1 #2 #3
  {
    \__nm_old_multicolumn { #1 } { #2 } { #3 }
    \int_compare:nNnT #1 > 1
      {
        \seq_gput_left:Nx \g__nm_multicolumn_cells_seq
          { \int_eval:n \c@iRow - \int_use:N \c@jCol }
        \seq_gput_left:Nn \g__nm_multicolumn_sizes_seq { #1 }
      }
    \int_gadd:Nn \c@jCol { #1 - 1 }
  }
```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArray}`. This command uses an optional argument like `\hdotsfor` but this argument is discarded (in `\hdotsfor`, this argument is used for fine tuning of the space between two consecutive dots). Tikz nodes are created for all the cells of the array, even the implicit cells of the `\Hdotsfor`.

This command must not be protected since it begins with `\multicolumn`.

```
\cs_new:Npn \__nm_Hdotsfor:
  {
    \multicolumn { 1 } { C } { }
    \__nm_Hdotsfor_i
  }
```

The command `\@@_Hdotsfor_i` is defined with the tools of xparse because it has an optionnal 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:`).

```
\bool_if:NTF \c__nm_draft_bool
  {
    \NewDocumentCommand \__nm_Hdotsfor_i { O { }  m }
      { \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } }
  }
  {
    \NewDocumentCommand \__nm_Hdotsfor_i { O { }  m }
      {
        \tl_gput_right:Nx \g__nm_Hdotsfor_lines_tl
          {
```

```
          \__nm_draw_Hdotsfor:nnn
            { \int_use:N \c@iRow }
            { \int_use:N \c@jCol }
            { #2 }
        }
      \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } }
    }
}
```

```
\cs_new_protected:Nn \__nm_draw_Hdotsfor:nnn
  {
    \bool_set_false:N \l__nm_initial_open_bool
    \bool_set_false:N \l__nm_final_open_bool
```

For the row, it's easy.

```
    \int_set:Nn \l__nm_initial_i_int { #1 }
    \int_set:Nn \l__nm_final_i_int { #1 }
```

For the column, it's a bit more complicated.

```
    \int_compare:nNnTF #2 = 1
      {
        \int_set:Nn \l__nm_initial_j_int 1
        \bool_set_true:N \l__nm_initial_open_bool
      }
      {
        \int_set:Nn \l_tmpa_int { #2 - 1 }
        \__nm_if_not_empty_cell:nnTF \l__nm_initial_i_int \l_tmpa_int
          { \int_set:Nn \l__nm_initial_j_int { #2 - 1 } }
          {
            \int_set:Nn \l__nm_initial_j_int {#2}
            \bool_set_true:N \l__nm_initial_open_bool
          }
      }
    \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
      {
        \int_set:Nn \l__nm_final_j_int { #2 + #3 - 1 }
        \bool_set_true:N \l__nm_final_open_bool
      }
      {
        \int_set:Nn \l_tmpa_int { #2 + #3 }
        \__nm_if_not_empty_cell:nnTF \l__nm_final_i_int \l_tmpa_int
          { \int_set:Nn \l__nm_final_j_int { #2 + #3 } }
          {
            \int_set:Nn \l__nm_final_j_int { #2 + #3 - 1 }
            \bool_set_true:N \l__nm_final_open_bool
          }
      }
    \bool_if:nT { \l__nm_initial_open_bool || \l__nm_final_open_bool }
      \__nm_create_extra_nodes:
    \__nm_actually_draw_Ldots:
```

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 { __nm _ dotted _ #1 - ##1 } { } }
  }
```

17.11 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 specification 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).[31]

```
\cs_new:Npn \__nm_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).

```
\cs_new_protected:Npn \__nm_line:nn #1 #2
  {
    \use:x
      {
        \__nm_line_i:nn
          { \__nm_double_int_eval:n #1 \q_stop }
          { \__nm_double_int_eval:n #2 \q_stop }
      }
  }
\cs_new_protected:Nn \__nm_line_i:nn
  {
    \bool_if:NF \c__nm_draft_bool
      {
        \dim_zero_new:N \g__nm_x_initial_dim
        \dim_zero_new:N \g__nm_y_initial_dim
        \dim_zero_new:N \g__nm_x_final_dim
        \dim_zero_new:N \g__nm_y_final_dim
        \bool_set_false:N \l__nm_initial_open_bool
        \bool_set_false:N \l__nm_final_open_bool
        \bool_if:nTF
          {
            \cs_if_exist_p:c { pgf@sh@ns@nm - \int_use:N \g__nm_env_int - #1 }
            &&
            \cs_if_exist_p:c { pgf@sh@ns@nm - \int_use:N \g__nm_env_int - #2 }
          }
          {
            \begin { tikzpicture }
              \path~(#1)~--~(#2)~node[at~start]~(i)~{}~node[at~end]~(f)~{} ;
              \tikz@parse@node \pgfutil@firstofone ( i )
              \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
              \dim_gset:Nn \g__nm_y_initial_dim \pgf@y
              \tikz@parse@node \pgfutil@firstofone ( f )
              \dim_gset:Nn \g__nm_x_final_dim \pgf@x
              \dim_gset:Nn \g__nm_y_final_dim \pgf@y
            \end { tikzpicture }
            \__nm_draw_tikz_line:
          }
          {
            \__nm_error:nnn { unknown~cell~for~line~in~code-after }
              { #1 } { #2 }
          }
      }
  }
```

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).

17.12 The commands to draw dotted lines to separate columns and rows

The command `\hdottedline` draws an horizontal dotted line to separate two rows. Similarly, the letter ":" in the preamble draws a vertical dotted line (the letter can be changed with the option

[31]Indeed, we want that the user may use the command `\line` in `code-after` with LaTeX counters in the arguments — with the command `\value`.

letter-for-dotted-lines). Both mechanisms write instructions in the code-after. The actual instructions in the code-after use the commands \@@_hdottedline:n and \@@_vdottedline:n.

We want the horizontal lines at the same position[32] as the line created by \hline (or \hdashline of arydshln). That's why we use a \noalign to insert a box with a \dotfill.

Some extensions, like the extension doc, do a redefinition of the command \dotfill of LaTeX. That's why we define a command \@@_dotfill: as we wish. We test whether we are in draft mode because, in this case, we don't draw the dotted lines.

```
\bool_if:NTF \c__nm_draft_bool
  { \cs_set_eq:NN \__nm_dotfill: \prg_do_nothing: }
  {
    \cs_set:Npn \__nm_dotfill:
      {
```

If the option small is used, we change the space between dots (we can't use \l_@@_inter_dots_dim which will be set after the construction of the array). We can't put the \bool_if:NT in the first argument of \hbox_to_wd:nn because \cleaders is a special TeX primitive.

```
        \bool_if:NT \l__nm_small_bool
          { \dim_set:Nn \l__nm_inter_dots_dim { 0.25 em } }
        \cleaders
        \hbox_to_wd:nn
          { \l__nm_inter_dots_dim }
          {
            \c_math_toggle_token
            \bool_if:NT \l__nm_small_bool \scriptstyle
            \hss . \hss
            \c_math_toggle_token
          }
        \hfill
      }
  }
```

This command must *not* be protected because it starts with \noalign.

```
\cs_new:Npn \__nm_hdottedline:
  {
    \noalign
      {
        \bool_gset_true:N \g__nm_extra_nodes_bool
        \cs_if_exist:cTF { __nm_width_ \int_use:N \g__nm_env_int }
          { \dim_set_eq:Nc \l_tmpa_dim { __nm_width_ \int_use:N \g__nm_env_int } }
          { \dim_set:Nn \l_tmpa_dim { 5 mm } }
        \hbox_overlap_right:n
          {
            \bool_if:nT
              {
                \l__nm_NiceArray_bool
                    &&
                ! \l__nm_exterior_arraycolsep_bool
                    &&
                \int_compare_p:nNn \l__nm_first_col_int > \c_zero_int
              }
              { \skip_horizontal:n { - \arraycolsep } }
            \hbox_to_wd:nn
              {
                \l_tmpa_dim + 2 \arraycolsep
                - \l__nm_left_margin_dim - \l__nm_right_margin_dim
              }
            \__nm_dotfill:
          }
      }
  }
```

[32]In fact, almost the same position because of the width of the line: the width of a dotted line is not the same as the width of a line created by \hline.

```
\cs_new_protected:Nn \__nm_vdottedline:n
  {
```

We should allow the letter "`:`" in the first position of the preamble but that would need a special programmation.

```
    \int_compare:nNnTF #1 = \c_zero_int
      { \__nm_error:n { Use~of~:~in~first~position } }
      {
        \bool_if:NF \c__nm_draft_bool
          {
            \dim_zero_new:N \g__nm_x_initial_dim
            \dim_zero_new:N \g__nm_y_initial_dim
            \dim_zero_new:N \g__nm_x_final_dim
            \dim_zero_new:N \g__nm_y_final_dim
            \bool_set_true:N \l__nm_initial_open_bool
            \bool_set_true:N \l__nm_final_open_bool
```

If a "`col`" node exists (if the array has been constructed with a fixed width of column), we use it.

```
            \cs_if_exist:cTF
              { pgf@sh@ns@nm -\int_use:N \g__nm_env_int - col - #1 }
              {
                \begin { tikzpicture } [ remember~picture ]
                  \tikz@parse@node\pgfutil@firstofone
                    ( col - #1 )
                  \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
                  \dim_gset:Nn \g__nm_x_final_dim \pgf@x
                  \dim_gset:Nn \g__nm_y_final_dim \pgf@y
                \end { tikzpicture }
                \dim_gset:Nn \g__nm_y_initial_dim { - \c_max_dim }
                \int_step_inline:nn \c@jCol
                  {
                    \begin { tikzpicture } [ remember~picture ]
                      \tikz@parse@node\pgfutil@firstofone
                        ( 1 - ##1 . north~east )
                      \dim_gset:Nn \g__nm_y_initial_dim
                        { \dim_max:nn \g__nm_y_initial_dim \pgf@y }
                    \end { tikzpicture }
                  }
              }
```

If not, we use the "large node".

```
              {
                \begin { tikzpicture } [ remember~picture ]
                  \tikz@parse@node\pgfutil@firstofone
                    ( 1 - #1 - large .north~east )
                  \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
                  \dim_gset:Nn \g__nm_y_initial_dim \pgf@y
                  \tikz@parse@node\pgfutil@firstofone
                    ( \int_use:N \c@iRow - #1 - large .south~east )
                  \dim_gset:Nn \g__nm_x_final_dim \pgf@x
                  \dim_gset:Nn \g__nm_y_final_dim \pgf@y
                \end { tikzpicture }
```

However, if the previous column was constructed with a letter `w`, we use the `w`-nodes (and we erase the previous computation of the x-value of the vertical dotted line).

```
                \cs_if_exist:cT
                  { pgf@sh@ns@nm -\int_use:N \g__nm_env_int - 1 - #1 - w }
                  {
                    \begin { tikzpicture } [ remember~picture ]
                      \tikz@parse@node\pgfutil@firstofone
                        ( 1 - #1 - w .north~east )
                      \dim_gset:Nn \g__nm_x_initial_dim \pgf@x
                      \tikz@parse@node\pgfutil@firstofone
                        ( \int_use:N \c@iRow - #1 - w .south~east )
                      \dim_gset:Nn \g__nm_x_final_dim \pgf@x
```

```
                    \end { tikzpicture }
                    \dim_gadd:Nn \g__nm_x_initial_dim \arraycolsep
                    \dim_gadd:Nn \g__nm_x_final_dim \arraycolsep
                  }
              }
            \__nm_draw_tikz_line:
          }
      }
  }
```

17.13 The vertical rules

We don't want that a vertical rule drawn by the specifier "|" extends in the eventual "first row" and "last row" of the array.
The natural way to do that would be to redefine the specifier "|" with \newcolumntype:

```
\newcolumntype { | }
  { ! { \int_compare:nNnF \c@iRow = \c_zero_int \vline } }
```

However, this code fails is 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 will do a redefinition of the macro \@arrayrule of array and this redefinition will add \@@_vline: instead of \vline to the preamble.
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 must be effective in each row and not once when the preamble is constructed).

```
\cs_new_protected:Npn \__nm_vline:
  {
    \int_compare:nNnTF \l__nm_first_col_int = \c_zero_int
      {
        \int_compare:nNnTF \c@jCol = \c_zero_int
          {
            \int_compare:nNnTF \l__nm_first_row_int = \c_zero_int
              {
                \int_compare:nNnF \c@iRow = \c_zero_int
                  {
                    \int_compare:nNnF \c@iRow = \l__nm_last_row_int
                    \__nm_vline_i:
                  }
              }
              {
                \int_compare:nNnF \c@iRow = \c_zero_int
                  {
                    \int_compare:nNnF \c@iRow = \l__nm_last_row_int
                    \__nm_vline_i:
                  }
              }
           }
           {
             \int_compare:nNnF \c@iRow = \c_zero_int
              {
                \int_compare:nNnF \c@iRow = \l__nm_last_row_int
                \__nm_vline_i:
              }
           }
      }
      {
        \int_compare:nNnTF \c@jCol = \c_zero_int
          {
            \int_compare:nNnF \c@iRow = { -1 }
              {
```

```
                \int_compare:nNnF \c@iRow = { \l__nm_last_row_int - 1 }
                  \__nm_vline_i:
              }
          }
          {
            \int_compare:nNnF \c@iRow = \c_zero_int
              {
                \int_compare:nNnF \c@iRow = \l__nm_last_row_int
                \__nm_vline_i:
              }
          }
      }
  }
```

If colortbl is loaded, the following macro will redefined (in a `\AtBeginDocument`) to take into account the color fixed by `\arrayrulecolor` of colortbl.

```
\cs_set_eq:NN \__nm_vline_i: \vline
```

17.14 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__nm_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__nm_block_auto_columns_width_bool
        \dim_gzero_new:N \g__nm_max_cell_width_dim
        \bool_set_true:N \l__nm_auto_columns_width_bool
      }
  }
```

```
\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
  {
    \int_gincr:N \g__nm_NiceMatrixBlock_int
    \dim_zero:N \l__nm_columns_width_dim
    \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
    \bool_if:NT \l__nm_block_auto_columns_width_bool
      {
        \cs_if_exist:cT { __nm_max_cell_width_ \int_use:N \g__nm_NiceMatrixBlock_int }
          {
            \dim_set:Nx \l__nm_columns_width_dim
              { \use:c { __nm_max_cell_width _ \int_use:N \g__nm_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__nm_block_auto_columns_width_bool
      {
        \iow_now:Nn \@mainaux \ExplSyntaxOn
        \iow_now:Nx \@mainaux
          {
```

```
            \cs_gset:cpn
              { __nm _ max _ cell _ width _ \int_use:N \g__nm_NiceMatrixBlock_int }
              { \dim_use:N \g__nm_max_cell_width_dim }
          }
        \iow_now:Nn \@mainaux \ExplSyntaxOff
      }
  }
```

17.15 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 }
```

The macro `\@@_create_extra_nodes:` must *not* be used in the `code-after` because the `code-after` is executed in a scope of `prefix name`.

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:Nn \__nm_create_extra_nodes:
  {
    \begin { tikzpicture } [ remember~picture , overlay ]
      \int_step_variable:nnNn \l__nm_first_row_int \g__nm_row_total_int \__nm_i:
        {
          \dim_zero_new:c { l__nm_row_\__nm_i: _min_dim }
          \dim_set_eq:cN { l__nm_row_\__nm_i: _min_dim } \c_max_dim
          \dim_zero_new:c { l__nm_row_\__nm_i: _max_dim }
          \dim_set:cn { l__nm_row_\__nm_i: _max_dim } { - \c_max_dim }
        }
      \int_step_variable:nnNn \l__nm_first_col_int \g__nm_col_total_int \__nm_j:
        {
          \dim_zero_new:c { l__nm_column_\__nm_j: _min_dim }
          \dim_set_eq:cN { l__nm_column_\__nm_j: _min_dim } \c_max_dim
          \dim_zero_new:c { l__nm_column_\__nm_j: _max_dim }
          \dim_set:cn { l__nm_column_\__nm_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__nm_first_row_int \g__nm_row_total_int \__nm_i:
        {
          \int_step_variable:nnNn
            \l__nm_first_col_int \g__nm_col_total_int \__nm_j:
```

Maybe the cell (i-j) is an implicit cell (that is to say a cell after implicit ampersands `&`). In this case, of course, we don't update the dimensions we want to compute.

```
            { \cs_if_exist:cT
                { pgf@sh@ns@nm - \int_use:N \g__nm_env_int - \__nm_i: - \__nm_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`.

```
                {
                  \tikz@parse@node \pgfutil@firstofone
                    ( nm - \int_use:N \g__nm_env_int
                         - \__nm_i: - \__nm_j: .south~west )
                  \dim_set:cn { l__nm_row_\__nm_i: _min_dim}
```

```
                  { \dim_min:vn { l__nm_row _ \__nm_i: _min_dim } \pgf@y }
                \seq_if_in:NxF \g__nm_multicolumn_cells_seq { \__nm_i: - \__nm_j: }
                  {
                    \dim_set:cn { l__nm_column _ \__nm_j: _min_dim}
                      { \dim_min:vn { l__nm_column _ \__nm_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`.

```
                \tikz@parse@node \pgfutil@firstofone
                  ( nm - \int_use:N \g__nm_env_int - \__nm_i: - \__nm_j: .north~east )
                \dim_set:cn { l__nm_row _ \__nm_i: _ max_dim }
                  { \dim_max:vn { l__nm_row _ \__nm_i: _ max_dim } \pgf@y }
                \seq_if_in:NxF \g__nm_multicolumn_cells_seq { \__nm_i: - \__nm_j: }
                  {
                    \dim_set:cn { l__nm_column _ \__nm_j: _ max_dim }
                      { \dim_max:vn { l__nm_column _ \__nm_j: _max_dim } \pgf@x }
                  }
              }
          }
      }
```

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" (after changing the value of `name-suffix`).

```
    \tikzset { name~suffix = -medium }
    \__nm_create_nodes:
```

For "large nodes", the exterior rows and columns don't interfer. That's why the loop over the rows will start at 1 and the loop over the columns will stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```
    \int_set:Nn \l__nm_first_row_int 1
    \int_set:Nn \l__nm_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 } \__nm_i:
      {
        \dim_set:cn { l__nm_row _ \__nm_i: _ min _ dim }
          {
            (
              \dim_use:c { l__nm_row _ \__nm_i: _ min _ dim } +
              \dim_use:c { l__nm_row _ \int_eval:n { \__nm_i: + 1 } _ max _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l__nm_row _ \int_eval:n { \__nm_i: + 1 } _ max _ dim }
          { l__nm_row_\__nm_i: _min_dim }
      }
    \int_step_variable:nNn { \c@jCol - 1 } \__nm_j:
      {
        \dim_set:cn { l__nm_column _ \__nm_j: _ max _ dim }
          {
            (
              \dim_use:c
                { l__nm_column _ \__nm_j: _ max _ dim } +
              \dim_use:c
                { l__nm_column _ \int_eval:n { \__nm_j: + 1 } _ min _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l__nm_column _ \int_eval:n { \__nm_j: + 1 } _ min _ dim }
          { l__nm_column _ \__nm_j: _ max _ dim }
      }
    \dim_sub:cn
```

```
        { l__nm_column _ 1 _ min _ dim }
        \l__nm_left_margin_dim
      \dim_add:cn
        { l__nm_column _ \int_use:N \c@jCol _ max _ dim }
        \l__nm_right_margin_dim
```

Now, we can actually create the “large nodes”.

```
      \tikzset { name~suffix = -large }
      \__nm_create_nodes:
      \end{tikzpicture}
```

When used once, the command `\@@_create_extra_nodes:` must become no-op (in the current TeX group). That’s why we put a nullification of the command.

```
      \cs_set:Npn \__nm_create_extra_nodes: { }
```

We can now compute the width of the array (used by `\hdottedline`).

```
      \begin { tikzpicture } [ remember~picture , overlay ]
        \tikz@parse@node \pgfutil@firstofone
          ( nm - \int_use:N \g__nm_env_int - 1 - 1 - large .north~west )
        \dim_gset:Nn \g_tmpa_dim \pgf@x
        \tikz@parse@node \pgfutil@firstofone
          ( nm - \int_use:N \g__nm_env_int - 1 -
                 \int_use:N \c@jCol - large .north~east )
        \dim_gset:Nn \g_tmpb_dim \pgf@x
      \end { tikzpicture }
      \iow_now:Nn \@mainaux \ExplSyntaxOn
      \iow_now:Nx \@mainaux
        {
          \cs_gset:cpn { __nm_width_ \int_use:N \g__nm_env_int }
            { \dim_eval:n { \g_tmpb_dim - \g_tmpa_dim } }
        }
      \iow_now:Nn \@mainaux \ExplSyntaxOff
  }
```

The control sequence `\@@_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.

```
\cs_new_protected:Nn \__nm_create_nodes:
  {
    \int_step_variable:nnNn \l__nm_first_row_int \g__nm_row_total_int \__nm_i:
      {
        \int_step_variable:nnNn \l__nm_first_col_int \g__nm_col_total_int \__nm_j:
```

We create two ponctual nodes for the extremities of a diagonal of the rectangular node we want to create. These nodes `(@@~south~west)` and `(@@~north~east)` are not available for the user of nicematrix. That’s why their names are independent of the row and the column. In the two nested loops, they will be overwritten until the last cell.

```
          {
            \coordinate ( __nm~south~west )
               at ( \dim_use:c { l__nm_column_ \__nm_j: _min_dim } ,
                    \dim_use:c { l__nm_row_ \__nm_i: _min_dim } ) ;
            \coordinate ( __nm~north~east )
               at ( \dim_use:c { l__nm_column_ \__nm_j: _max_dim } ,
                    \dim_use:c { l__nm_row_ \__nm_i: _max_dim } ) ;
```

We can eventually draw the rectangular node for the cell (`\@@_i`-`\@@_j`). This node is created with the Tikz library fit. Don’t forget that the Tikz option `name suffix` has been set to `-medium` or `-large`.

```
            \node
              [
                node~contents = { } ,
```

```
                fit = ( __nm~south~west ) ( __nm~north~east )  ,
                inner~sep = \c_zero_dim ,
                name = nm - \int_use:N \g__nm_env_int - \__nm_i: - \__nm_j: ,
                alias =
                  \str_if_empty:NF \l__nm_name_str
                    { \l__nm_name_str - \__nm_i: - \__nm_j: }
              ]
              ;
          }
      }
```

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__nm_multicolumn_cells_seq
      \g__nm_multicolumn_sizes_seq
      \__nm_node_for_multicolumn:nn
  }
```

```
\cs_new_protected:Npn \__nm_extract_coords: #1 - #2 \q_stop
  {
    \cs_set:Npn \__nm_i: { #1 }
    \cs_set:Npn \__nm_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:Nn \__nm_node_for_multicolumn:nn
  {
    \__nm_extract_coords: #1 \q_stop
    \coordinate ( __nm~south~west ) at
      (
        \dim_use:c { l__nm_column _ \__nm_j: _ min _ dim } ,
        \dim_use:c { l__nm_row _ \__nm_i: _ min _ dim }
      ) ;
    \coordinate ( __nm~north~east ) at
      (
        \dim_use:c { l__nm_column _ \int_eval:n { \__nm_j: + #2 - 1 } _ max _ dim} ,
        \dim_use:c { l__nm_row _ \__nm_i: _ max _ dim }
      ) ;
    \node
      [
        node~contents = { } ,
        fit = ( __nm~south~west ) ( __nm~north~east ) ,
        inner~sep = \c_zero_dim ,
        name = nm - \int_use:N \g__nm_env_int - \__nm_i: - \__nm_j: ,
        alias =
          \str_if_empty:NF \l__nm_name_str
            { \l__nm_name_str - \__nm_i: - \__nm_j: }
      ]
      ;
  }
```

17.16 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 `\NewExpandableDocumentCommand` of `xparse` because it has an optionnal argument between `<` and `>` (for TeX instructions put before the math mode of the label) and because it must be expandable since it reduces (in the case of a block of only one row) to a command `\multicolumn`.

```
\NewExpandableDocumentCommand \__nm_Block: { m D < > { } m }
  { \__nm_Block_i #1 \q_stop { #2 } { #3 } }
```

The first argument of `\@@_Block:` (which is required) 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 \__nm_Block_i #1-#2 \q_stop { \__nm_Block_ii:nnnn { #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` are the tokens to put before the math mode and `#4` is the label of the block. The following command must *not* be protected because it contains a command `\multicolumn` (in the case of a block of only one row).

```
\cs_new:Npn \__nm_Block_ii:nnnn #1 #2 #3 #4
  {
```

In the case of a block of only one row, we use a `\multicolumn` and not the general technique because, in this case, we want the label perfectly aligned with the base line of that row of the array.

```
    \int_compare:nNnTF { #1 } = 1
      {
        \multicolumn { #2 } { C } { \hbox:n { #3 $#4$ } }
        \__nm_gobble_ampersands:n { #2 - 1 }
      }
      { \__nm_Block_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
  }
```

The command `\@@_Block_iii:nnnn` is for the case of a block of n rows with $n > 1$.

```
\cs_new_protected:Npn \__nm_Block_iii:nnnn #1 #2 #3 #4
  {
    \bool_gset_true:N \g__nm_extra_nodes_bool
```

We write an instruction in the `code-after`. We write the instruction in the beginning of the `code-after` (the `left` in `\tl_gput_left:Nx`) because we want the Tikz nodes corresponding of the block created *before* potential instructions written by the user in the `code-after` (these instructions may use the Tikz node of the created block).

```
    \tl_gput_left:Nx \g__nm_code_after_tl
      {
        \__nm_Block_iv:nnnnn
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \int_eval:n { \c@iRow + #1 - 1 } }
          { \int_eval:n { \c@jCol + #2 - 1 } }
          \exp_not:n { { #3 $ #4 $ } }
      }
  }
```

The command `\@@_gobble_ampersands:n` will gobble n ampersands (and also the spaces) where n is the argument of the command. This command is fully expandable and we need this feature.

```
\group_begin:
  \char_set_catcode_letter:N \&
  \cs_new:Npn \__nm_gobble_ampersands:n #1
    {
      \int_compare:nNnT { #1 } > 0
        {
          \peek_charcode_remove_ignore_spaces:NT &
            { \__nm_gobble_ampersands:n { #1 - 1 } }
        }
    }
\group_end:
```

The following command `\@@_Block_iii:nnnnn` will be used in the `code-after`. It's necessary to create two Tikz nodes because we want the label `#5` really drawn in the *center* of the node.

```
\cs_new_protected:Npn \__nm_Block_iv:nnnnn #1 #2 #3 #4 #5
  {
```

```
  \bool_if:nTF
   {
       \int_compare_p:nNn { #3 } > \c@iRow
    || \int_compare_p:nNn { #4 } > \c@jCol
   }
   { \msg_error:nnnn { nicematrix } { Block~too~large } { #1 } { #2 } }
   {
    \begin{tikzpicture}
    \node
     [
      fit = ( #1 - #2 - medium . north~west )
            ( #3 - #4 - medium . south~east ) ,
      inner~sep = 0 pt ,
     ]
```

We don’t forget the name of the node because the user may wish to use it.

```
     (#1-#2) { } ;
    \node at (#1-#2.center) { #5 } ;
  \end{tikzpicture}
   }
 }
```

17.17 How to draw the dotted lines transparently

```
\cs_set_protected:Npn \__nm_renew_matrix:
 {
  \RenewDocumentEnvironment { pmatrix } { }
   { \pNiceMatrix }
   { \endpNiceMatrix }
  \RenewDocumentEnvironment { vmatrix } { }
   { \vNiceMatrix }
   { \endvNiceMatrix }
  \RenewDocumentEnvironment { Vmatrix } { }
   { \VNiceMatrix }
   { \endVNiceMatrix }
  \RenewDocumentEnvironment { bmatrix } { }
   { \bNiceMatrix }
   { \endbNiceMatrix }
  \RenewDocumentEnvironment { Bmatrix } { }
   { \BNiceMatrix }
   { \endBNiceMatrix }
 }
```

17.18 Automatic arrays

```
\cs_new_protected:Npn \__nm_set_size:n #1-#2 \q_stop
 {
  \int_set:Nn \l__nm_nb_rows_int { #1 }
  \int_set:Nn \l__nm_nb_cols_int { #2 }
 }
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
 {
  \int_zero_new:N \l__nm_nb_rows_int
  \int_zero_new:N \l__nm_nb_cols_int
  \__nm_set_size:n #4 \q_stop
  \begin { NiceArrayWithDelims } { #1 } { #2 }
   { * { \l__nm_nb_cols_int } { C } } [ #3 , #5 , #7 ]
  \int_compare:nNnT \l__nm_first_row_int = \c_zero_int
   {
    \int_compare:nNnT \l__nm_first_col_int = \c_zero_int { & }
    \prg_replicate:nn { \l__nm_nb_cols_int - 1 } { & }
    \int_compare:nNnT \l__nm_last_col_int > { -1 } { & } \\
   }
```

```
    \prg_replicate:nn \l__nm_nb_rows_int
      {
        \int_compare:nNnT \l__nm_first_col_int = \c_zero_int { & }
```

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__nm_nb_cols_int - 1 } { { } #6 & } #6
        \int_compare:nNnT \l__nm_last_col_int > { -1 } { & } \\
      }
    \int_compare:nNnT \l__nm_last_row_int > { -2 }
      {
        \int_compare:nNnT \l__nm_first_col_int = \c_zero_int { & }
        \prg_replicate:nn { \l__nm_nb_cols_int - 1 } { & }
        \int_compare:nNnT \l__nm_last_col_int > { -1 } { & } \\
      }
    \end { NiceArrayWithDelims }
  }
```

```
\cs_set_protected:Npn \__nm_define_com:nnn #1 #2 #3
  {
    \cs_set_protected:cpn { #1 AutoNiceMatrix }
      {
        \str_gset:Nx \g__nm_type_env_str
          { command ~ \c_backslash_str #1 AutoNiceMatrix }
        \AutoNiceMatrixWithDelims { #2 } { #3 }
      }
  }
%    \end{document}
%    \begin{macrocode}
\__nm_define_com:nnn p ( )
\__nm_define_com:nnn b [ ]
\__nm_define_com:nnn v | |
\__nm_define_com:nnn V \| \|
\__nm_define_com:nnn B \{ \}
```

17.19 We process the options

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__nm_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    renew-matrix .code:n = \__nm_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    transparent .meta:n = { renew-dots , renew-matrix } ,
    transparent .value_forbidden:n = true,
  }
\ProcessKeysOptions { NiceMatrix / Package }
```

17.20 Error messages of the package

```
\__nm_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~Tikz~node~doesn't~exist.\\
    If~you~go~on~this~command~will~be~ignored.
  }
```

```
\__nm_msg_new:nn { last-col~non~empty~for~NiceArray }
```

```
  {
    In~the~\g__nm_type_env_str,~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).
  }
\__nm_msg_new:nn { last-col~empty~for~NiceMatrix }
  {
    In~the~\g__nm_type_env_str, you~can't~use~the~option~
    'last-col'~without~value.~You~must~give~the~number~of~that~last~column.\\
    If~you~go~on~this~option~will~be~ignored.
  }
\__nm_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.
  }
\__nm_msg_new:nn { Impossible~line }
  {
    A~dotted~line~can't~be~drawn~because~you~have~not~put~
    all~the~ampersands~required~on~the~row~#1.\\
    If~you~go~on,~this~dotted~line~will~be~ignored.
  }
\__nm_msg_new:nn { Wrong~last~row }
  {
    You~have~used~'last-row=\int_use:N \l__nm_last_row_int'~but~your~
    \g__nm_type_env_str\ 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).
  }
\__nm_msg_new:nn { Yet~in~env }
  {
    Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~nested.\\
    This~error~is~fatal.
  }
\__nm_msg_new:nn { Outside~math~mode }
  {
    The~\g__nm_type_env_str\ can~be~used~only~in~math~mode~
    (and~not~in~\token_to_str:N \vcenter).\\
    This~error~is~fatal.
  }
\__nm_msg_new:nn { Option~Transparent~suppressed }
  {
    The~option~'Transparent'~has~been~renamed~'transparent'.\\
    However,~you~can~go~on~for~this~time.
  }
\__nm_msg_new:nn { Option~RenewMatrix~suppressed }
  {
    The~option~'RenewMatrix'~has~been~renamed~'renew-matrix'.\\
    However,~you~can~go~on~for~this~time.
  }
\__nm_msg_new:nn { Bad~value~for~letter~for~dotted~lines }
  {
    The~value~of~key~'\tl_use:N\l_keys_key_tl'~must~be~of~length~1.\\
    If~you~go~on,~it~will~be~ignored.
  }
\__nm_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
```

```
  {
    The~key~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~command~
    \token_to_str:N \NiceMatrixOptions. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    allow-duplicate-names,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    exterior-arraycolsep,~
    hlines,~
    left-margin,~
    letter-for-dotted-lines,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots,~
    renew-matrix,~
    right-margin,~
    small~
    and~transparent
  }
\__nm_msg_new:nnn { Unknown~option~for~NiceArray }
  {
    The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
    \{NiceArray\}. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~options,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    b,~
    c,~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    columns-width,~
    create-extra-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    last-col,~
    last-row,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots,~
    right-margin,~
    small~
    and~t.
  }
```

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`).

```
\__nm_msg_new:nnn { Unknown~option~for~NiceMatrix }
```

```
  {
    The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~
    \g__nm_type_env_str. \\
    If~you~go~on,~it~will~be~ignored. \\
    For~a~list~of~the~available~options,~type~H~<return>.
  }
  {
    The~available~options~are~(in~alphabetic~order):~
    code-after,~
    code-for-first-col,~
    code-for-first-row,~
    code-for-last-col,~
    code-for-last-row,~
    columns-width,~
    create-extra-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    last-col,~
    last-row,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots,~
    right-margin~
    and~small.
  }
```

The following message should be changed because, normally, there can be any longer artefact in the environments of amsmath.

```
\__nm_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~use~nicematrix~inside~some~
    environments~of~amsmath,~this~error~may~be~an~artefact.~In~this~case,~
    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__nm_names_seq { ,~ } { ,~ } { ~and~ }.
  }
\__nm_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.
  }
\__nm_msg_new:nn { Zero~row }
  {
    There~is~a~problem.~Maybe~your~\g__nm_type_env_str\ is~empty.~
    Maybe~you~have~used~l,~c~and~r~instead~of~L,~C~and~R~in~the~preamble~
    of~your~environment. \\
    If~you~go~on,~the~result~may~be~incorrect.
  }
\__nm_msg_new:nn { Use~of~:~in~first~position }
  {
    You~can't~use~the~column~specifier~'\l__nm_letter_for_dotted_lines_str'~in~the~
    first~position~of~the~preamble~of~the~\g__nm_type_env_str. \\
```

```
    If~you~go~on,~this~dotted~line~will~be~ignored.
  }
```

17.21 Obsolete environments

```
\__nm_msg_new:nn { Obsolete~environment }
  {
    The~environment~\{\@currenvir\}~is~obsolete~and~will~be~deleted~in~future~version.~
    We~should~use~#1~instead.
  }
\NewDocumentEnvironment { pNiceArrayC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \pNiceArray
  }
  { \endpNiceArray }
\NewDocumentEnvironment { bNiceArrayC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \bNiceArray
  }
  { \endbNiceArray }
\NewDocumentEnvironment { BNiceArrayC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \BNiceArray
  }
  { \endBNiceArray }
\NewDocumentEnvironment { vNiceArrayC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \vNiceArray
  }
  { \endvNiceArray }
\NewDocumentEnvironment { VNiceArrayC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \VNiceArray
  }
  { \endVNiceArray }
\NewDocumentEnvironment { pNiceArrayRC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \pNiceArray
  }
  { \endpNiceArray }
\NewDocumentEnvironment { bNiceArrayRC } { }
  {
```

```
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \bNiceArray
  }
  { \endbNiceArray }
\NewDocumentEnvironment { BNiceArrayRC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \BNiceArray
  }
  { \endBNiceArray }
\NewDocumentEnvironment { vNiceArrayRC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \bool_set_true:N \l__nm_last_col_bool
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \vNiceArray
  }
  { \endvNiceArray }
\NewDocumentEnvironment { VNiceArrayRC } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \VNiceArray
  }
  { \endVNiceArray }
\NewDocumentEnvironment { NiceArrayCwithDelims } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~option~'last-col' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \NiceArrayWithDelims
  }
  { \endNiceArrayWithDelims }
\NewDocumentEnvironment { NiceArrayRCwithDelims } { }
  {
    \msg_info:nnn { nicematrix } { Obsolete~environment }
      { the~options~'last-col'~and~'first-row' }
    \int_set:Nn \l__nm_last_col_int \c_zero_dim
    \int_set:Nn \l__nm_first_row_int \c_zero_int
    \NiceArrayWithDelims
  }
  { \endNiceArrayWithDelims }
```

18 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[33], Tikz externalization is now deactivated in the environments of the extension nicematrix.[34]

Changes between version 2.1 and 2.1.2

Option `draft`: with this option, the dotted lines are not drawn (quicker).

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`).

$$\begin{array}{cc} \begin{array}{ccc} & C_j & \end{array} & \\ \begin{pmatrix} 0 & \vdots & 0 \\ & a & \cdots \\ 0 & & 0 \end{pmatrix} & \begin{array}{c} \\ L_i \\ \\ \end{array} \end{array}$$

[33] cf. `tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package`

[34] 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 2.1.3 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`.
A warning is issued when the `draft` mode is used. In this case, the dotted lines are not drawn.

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 (except 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[35], 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).

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

`\&` 2224
`\\` 2289, 2295, 2301, 2334, 2340, 2347, 2353, 2359, 2372, 2378, 2383, 2388, 2393, 2399, 2400, 2425, 2426, 2459, 2460, 2494, 2495, 2510, 2516
`\{` 984, 2320, 2333, 2372, 2425, 2521
`\}` 984, 2320, 2333, 2372, 2425, 2521
`\|` 1000, 2319
`\ ` 2365, 2366, 2377, 2508

A

`\array` 438
`\arraycolsep` 165, 167, 169, 470, 619, 620, 679, 682, 690, 694, 719, 726, 758, 796, 798, 812, 881, 919, 1871, 1874, 1937, 1938
`\arrayrulewidth` 452, 453
`\arraystretch` 469
`\AtBeginDocument` 64, 92
`\AutoNiceMatrixWithDelims` 2278, 2311

B

`\begin` 1009, 1057, 1300, 1323, 1338, 1655, 1817, 1897, 1907, 1916, 1929, 2035, 2121, 2243, 2283, 2315
`\bgroup` 354
`\Block` 517
`\BNiceArray` 2545, 2588
`\bNiceArray` 2537, 2579
`\BNiceMatrix` 2270

[35] cf. `tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize`

\bNiceMatrix 2267
bool commands:
\bool_do_until:Nn 1171, 1231
\bool_gset_eq:NN 472
\bool_gset_false:N 562
\bool_gset_true:N 550, 888, 1857, 2212
\bool_if:NTF 36, 100, 292, 414, 430, 446, 467, 475, 518, 560, 579, 588, 591, 617, 648, 650, 657, 659, 735, 761, 809, 823, 894, 1054, 1097, 1115, 1124, 1131, 1195, 1223, 1255, 1284, 1292, 1318, 1330, 1355, 1362, 1377, 1392, 1394, 1406, 1410, 1425, 1440, 1442, 1454, 1458, 1463, 1471, 1476, 1482, 1486, 1501, 1505, 1510, 1514, 1543, 1547, 1552, 1556, 1600, 1602, 1613, 1635, 1636, 1637, 1682, 1688, 1694, 1700, 1706, 1731, 1802, 1834, 1839, 1846, 1886, 2009, 2019
\bool_if:nTF 545, 670, 683, 700, 824, 895, 1093, 1291, 1681, 1687, 1693, 1699, 1705, 1783, 1810, 1863, 2236
\bool_new:N 11, 27, 54, 55, 62, 63, 87, 90, 91, 132, 133, 135, 136, 137, 139, 140, 1994
\bool_set:Nn 1467, 1480
\bool_set_false:N 1030, 1044, 1129, 1130, 1170, 1175, 1230, 1235, 1353, 1404, 1452, 1499, 1541, 1751, 1752, 1808, 1809
\bool_set_true:N 12, 29, 32, 68, 95, 134, 179, 230, 580, 597, 957, 1052, 1179, 1185, 1191, 1199, 1203, 1227, 1239, 1245, 1251, 1259, 1264, 1288, 1758, 1766, 1772, 1780, 1892, 1893, 1999, 2001, 2595
\l_tmpa_bool 1030, 1044, 1052, 1054, 1467, 1486
\l_tmpb_bool 1480, 1482
box commands:
\box_clear_new:N 643
\box_dp:N . . 312, 324, 374, 486, 492, 496, 806
\box_ht:N . . 314, 319, 322, 488, 490, 494, 805
\box_move_down:nn 375, 776
\box_move_up:nn 389, 767
\box_set_dp:Nn 806
\box_set_ht:Nn 805
\box_use:N 355, 373, 389, 873, 951
\box_use_drop:N 784, 797, 807
\box_wd:N 331, 379, 632, 640, 846, 917
\l_tmpa_box 290, 312, 314, 319, 322, 324, 331, 355, 364, 373, 625, 632, 633, 640, 787, 805, 806, 807, 821, 846, 873, 892, 917, 951
\l_tmpb_box 372, 374, 379, 389

C

\Cdots 509
\cdots 521, 1669
Cdots internal commands:
__nm_Cdots 509, 521, 1685
cdots internal commands:
__nm_cdots 1669, 1688
char commands:
\char_set_catcode_letter:N 2224
\cleaders 1841
\coordinate 383, 388, 681, 698, 709, 721, 2144, 2147, 2176, 2181
\crcr 676
cs commands:
\cs_generate_variant:Nn 358, 1315, 2031, 2032
\cs_gset:Npn 1102, 1109, 2024, 2133
\cs_gset:Npx 1675
\cs_gset_eq:NN 113, 479
\cs_if_exist:NTF 460, 463, 581, 584, 600, 607, 1031, 1045, 1090, 1160, 1161, 1320, 1332, 1364, 1379, 1412, 1427, 1858, 1894, 1926, 2011, 2054
\cs_if_exist_p:N 1812, 1814
\cs_if_free:NTF 1035, 1215, 1276, 1351, 1402, 1450, 1497, 1539
\cs_new:Npn 441, 1715, 1726, 1789, 1853, 2200, 2201, 2225
\cs_new_protected:Nn 280, 300, 326, 359, 1078, 1087, 1158, 1163, 1294, 1316, 1349, 1358, 1400, 1448, 1495, 1537, 1579, 1673, 1709, 1749, 1800, 1881, 2033, 2138, 2173
\cs_new_protected:Npn . . . 18, 19, 20, 21, 22, 23, 24, 25, 56, 116, 307, 417, 428, 443, 458, 1003, 1791, 1945, 2168, 2210, 2234, 2273
\cs_set:Npn 435, 469, 473, 497, 1165, 1205, 1266, 1787, 1837, 2120, 2170, 2171
\cs_set_eq:NN 104, 432, 433, 434, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 520, 521, 522, 523, 524, 525, 526, 533, 534, 535, 537, 1151, 1668, 1669, 1670, 1671, 1672, 1714, 1835, 1993
\cs_set_protected:Npn 69, 98, 415, 590, 2255, 2305, 2307

D

\Ddots 511
\ddots 523, 1671
Ddots internal commands:
__nm_Ddots 511, 523, 1697
ddots internal commands:
__nm_ddots 1671, 1700
\DeclareOption 12, 13
dim commands:
\dim_abs:n 1071
\dim_add:Nn 2114
\dim_compare:nNnTF 1070, 1598
\dim_compare_p:nNn . . . 548, 673, 1468, 1481
\dim_eval:n 2134
\dim_gadd:Nn 1396, 1397, 1639, 1647, 1662, 1663, 1937, 1938
\dim_gset:Nn 311, 313, 318, 321, 323, 330, 486, 488, 490, 492, 494, 496, 619, 620, 632, 640, 689, 693, 842, 913, 1066, 1068, 1305, 1306, 1311, 1312, 1326, 1344, 1484, 1525, 1567, 1820, 1821, 1823, 1824, 1900, 1901, 1902, 1904, 1910, 1919, 1920, 1923, 1924, 1932, 1935, 2124, 2128
\dim_gset_eq:NN 303, 1393, 1395, 1441, 1443, 1489
\dim_gzero:N 304, 305
\dim_gzero_new:N 485, 487, 489, 491, 493, 495, 589, 615, 616, 1296, 1297, 1298, 1299, 2000
\dim_max:nn 312, 314, 319, 322, 324, 331, 844, 915, 1486, 1911, 2032, 2070, 2074
\dim_min:nn 1486, 2031, 2061, 2065
\dim_new:N 52, 73, 75, 141, 142, 143, 144, 145, 146

\dim_ratio:nn 1529, 1571, 1605, 1609, 1616, 1620, 1626, 1631, 1642, 1650
\dim_set:Nn 74, 76, 180, 235, 358, 374, 470, 746, 752, 1133, 1134, 1519, 1521, 1561, 1563, 1582, 1623, 1628, 1840, 1860, 2013, 2041, 2048, 2060, 2064, 2069, 2073, 2085, 2098
\dim_set_eq:NN 624, 635, 1859, 2039, 2046, 2093, 2108
\dim_sub:Nn 2111
\dim_use:N 1588, 1589, 1592, 1593, 2026, 2088, 2089, 2101, 2103, 2145, 2146, 2148, 2149, 2178, 2179, 2183, 2184
\dim_zero:N 749, 755, 2007
\dim_zero_new:N 1119, 1120, 1121, 1122, 1581, 1804, 1805, 1806, 1807, 1888, 1889, 1890, 1891, 2038, 2040, 2045, 2047
\c_max_dim 1904, 2039, 2041, 2046, 2048
\g_tmpa_dim 689, 693, 696, 707, 718, 1066, 1071, 2124, 2134
\l_tmpa_dim 374, 375, 389, 746, 749, 768, 793, 805, 1623, 1662, 1859, 1860, 1874
\g_tmpb_dim 1068, 1071, 2128, 2134
\l_tmpb_dim 752, 755, 778, 800, 806, 1628, 1663
\c_zero_dim 336, 337, 407, 548, 624, 635, 673, 853, 854, 931, 932, 1598, 2154, 2190, 2528, 2536, 2544, 2552, 2560, 2568, 2577, 2586, 2604, 2613, 2621
\dots 525

E

\egroup 356
else commands:
\else: 58
\end 1020, 1069, 1313, 1327, 1345, 1665, 1825, 1903, 1912, 1925, 1936, 2119, 2129, 2252, 2303, 2314
\endarray 728
\endBNiceArray 2547, 2590
\endbNiceArray 2539, 2581
\endBNiceMatrix 2271
\endbNiceMatrix 2268
\endNiceArrayWithDelims 962, 970, 978, 986, 994, 1002, 2616, 2625
\endpNiceArray 2531, 2572
\endpNiceMatrix 2259
\endVNiceArray 2563, 2608
\endvNiceArray 2555, 2599
\endVNiceMatrix 2265
\endvNiceMatrix 2262
\everycr 483, 499
exp commands:
\exp_after:wN 120
\exp_args:NV 540, 667
\exp_not:n 2220
\ExplSyntaxOff 1113, 2028, 2136
\ExplSyntaxOn 1099, 2021, 2130

F

\fi 189
fi commands:
\fi: 60
fp commands:
\fp_to_dim:n 1584

G

group commands:
\group_begin: 102, 623, 1089, 2223
\group_end: 107, 641, 1154, 2233

H

\halign 501, 503
hbox commands:
\hbox:n 43, 45, 47, 385, 794, 2205
\hbox_overlap_left:n 848
\hbox_overlap_right:n 920, 1861
\hbox_set:Nn 372, 625, 633, 787
\hbox_set:Nw 290, 364, 663, 821, 892
\hbox_set_end: 329, 370, 732, 840, 911
\hbox_to_wd:nn 379, 1842, 1872
\Hdotsfor 515
\hdotsfor 526
\hdottedline 513
\hfil 381, 537
\hfill 1850
\hrule 452
\Hspace 514
\hspace 1712
\hss 537, 1847

I

\ialign 473, 497
\Iddots 512
\iddots 38, 524, 1672
Iddots internal commands:
__nm_Iddots 512, 524, 1703
iddots internal commands:
__nm_iddots 1672, 1706
if commands:
\if_mode_math: 58
\ifstandalone 584
int commands:
\int_add:Nn 1173, 1174, 1257, 1258
\int_compare:nNnTF 283, 285, 293, 294, 296, 309, 316, 448, 450, 551, 595, 645, 654, 677, 733, 737, 744, 750, 756, 763, 772, 1013, 1048, 1080, 1095, 1176, 1178, 1182, 1184, 1188, 1190, 1236, 1238, 1242, 1244, 1248, 1250, 1517, 1559, 1718, 1755, 1769, 1883, 1947, 1949, 1951, 1953, 1955, 1960, 1962, 1968, 1970, 1976, 1978, 1980, 1985, 1987, 2203, 2227, 2285, 2287, 2289, 2293, 2295, 2297, 2299, 2301
\int_compare:nTF 241
\int_compare_p:nNn 826, 829, 831, 897, 900, 902, 1869, 2238, 2239
\int_eval:n 1015, 1721, 1790, 2089, 2093, 2104, 2108, 2183, 2218, 2219
\int_gadd:Nn 1724
\int_gdecr:N 1093
\int_gincr:N 282, 302, 587, 706, 717, 889, 2006
\int_gset:Nn 288, 530, 561, 699, 890
\int_gset_eq:NN 553, 740, 1092, 1094, 1160, 1161
\int_gsub:Nn 1096
\int_gzero:N 445
\int_gzero_new:N 462, 465, 531, 532, 559
\int_incr:N 1516, 1558
\int_max:nn 289, 891

\int_new:N 50, 51, 79, 80, 81, 83, 85, 88
\int_set:Nn 82, 84, 86, 89, 256, 602, 609, 1166, 1167, 1168, 1169, 1604, 1608, 1615, 1619, 1633, 1753, 1754, 1757, 1761, 1763, 1765, 1771, 1775, 1777, 1779, 2081, 2082, 2275, 2276, 2528, 2536, 2544, 2552, 2560, 2568, 2569, 2577, 2578, 2586, 2587, 2596, 2604, 2605, 2613, 2621, 2622
\int_set_eq:NN 461, 464
\int_step_inline:nn 1905
\int_step_inline:nnn 1656, 1786
\int_step_variable:nNn 2083, 2096
\int_step_variable:nnNn 2036, 2043, 2050, 2052, 2140, 2142
\int_sub:Nn 1197, 1198, 1233, 1234
\int_use:N 343, 344, 345, 350, 351, 397, 398, 399, 404, 405, 423, 424, 555, 600, 603, 681, 698, 711, 712, 723, 724, 862, 863, 869, 940, 941, 942, 947, 948, 1032, 1038, 1039, 1040, 1046, 1049, 1061, 1062, 1063, 1102, 1103, 1110, 1148, 1208, 1209, 1218, 1219, 1220, 1226, 1269, 1270, 1279, 1280, 1281, 1287, 1302, 1303, 1304, 1308, 1309, 1310, 1321, 1325, 1334, 1335, 1341, 1342, 1367, 1368, 1369, 1382, 1383, 1384, 1415, 1416, 1417, 1430, 1431, 1432, 1676, 1677, 1721, 1742, 1743, 1812, 1814, 1858, 1859, 1895, 1922, 1927, 1934, 2011, 2014, 2025, 2055, 2058, 2068, 2115, 2123, 2126, 2127, 2133, 2155, 2191, 2216, 2217, 2364, 2365, 2366
\int_zero:N 193, 194, 266, 271, 274, 275
\int_zero_new:N 1117, 1118, 1125, 1126, 1127, 1128, 2280, 2281
\c_one_int 241, 283, 285, 316, 1096, 1354, 1405, 1453, 1500, 1517, 1559
\g_tmpa_int 699, 706, 712, 717, 724
\l_tmpa_int 1604, 1608, 1615, 1619, 1643, 1651, 1656, 1761, 1762, 1775, 1776
\l_tmpb_int 1633, 1645, 1653
\c_zero_int 293, 294, 309, 645, 744, 756, 763, 826, 897, 1080, 1354, 1405, 1453, 1869, 1883, 1947, 1949, 1951, 1953, 1960, 1968, 1976, 1985, 2285, 2287, 2293, 2299, 2569, 2578, 2587, 2596, 2605, 2622
iow commands:
\iow_now:Nn 1099, 1100, 1107, 1113, 2021, 2022, 2028, 2130, 2131, 2136

K

\kern 47
keys commands:
\keys_define:nn 147, 175, 198, 220, 252, 259, 269, 1995, 2321
\l_keys_key_tl 2393, 2398, 2424, 2458
\keys_set:nn 251, 592, 593, 1008, 2008
\l_keys_value_tl 2342, 2489

L

\Ldots 508
\ldots 520, 1668
Ldots internal commands:
__nm_Ldots 508, 520, 525, 1679
ldots internal commands:
__nm_ldots 1668, 1682
\left 628, 637, 790
\line 1151, 2333
\lineskip 747, 753

M

\makebox 373
math commands:
\c_math_toggle_token 291, 328, 627, 629, 636, 638, 666, 729, 789, 803, 822, 839, 893, 910, 1845, 1848
\mathinner 40
\mkern 42, 44, 46, 47
msg commands:
\msg_error:nn 18
\msg_error:nnn 19, 1225, 1286
\msg_error:nnnn 20, 2241
\msg_fatal:nn 21, 22
\msg_info:nnn . . . 2526, 2534, 2542, 2550, 2558, 2566, 2575, 2584, 2593, 2602, 2611, 2619
\msg_new:nnn 23
\msg_new:nnnn 24
\msg_redirect_name:nnn 26
\msg_warning:nn 37
\multicolumn . . 516, 1714, 1728, 1734, 1746, 2205
\myfiledate 8
\myfileversion 9

N

\newcolumntype 361, 505, 506, 507, 540
\NewDocumentCommand 250, 1679, 1685, 1691, 1697, 1703, 1733, 1737, 2278
\NewDocumentEnvironment 570, 955, 963, 971, 979, 987, 995, 1005, 2004, 2524, 2532, 2540, 2548, 2556, 2564, 2573, 2582, 2591, 2600, 2609, 2617
\NewExpandableDocumentCommand 2198
\NiceArrayWithDelims 960, 968, 976, 984, 992, 1000, 2614, 2623
\NiceMatrixOptions 250, 2399
nm internal commands:
__nm_actualization_for_first_and_-last_row: 307, 332, 841, 912
__nm_actually_draw_Ldots: 1355, 1358, 1785
__nm_adapt_S_column: 98, 113, 577
__nm_add_to_empty_cells: 1673, 1683, 1689, 1695, 1701, 1707, 1711
__nm_adjust_with_col_nodes: 1316, 1391, 1439
__nm_after_array: 814, 1078
__nm_after_array_i: 1081, 1087
__nm_array: 428, 667
\l__nm_auto_columns_width_bool 137, 179, 547, 672, 685, 2001
__nm_begin_of_row: 286, 300, 820
__nm_Block: 517, 2198
\l__nm_block_auto_columns_width_bool 588, 686, 1994, 1999, 2009, 2019
__nm_Block_i 2199, 2200
__nm_Block_ii:nnnn 2200, 2201
__nm_Block_iii:nnnn 2208, 2210
__nm_Block_iv:nnnnn 2215, 2234
\g__nm_Cdots_lines_tl 563, 1140
__nm_Cell: 123, 280, 365, 505, 506, 507

\g__nm_code_after_tl 78, 191, 554, 1152, 1153, 2213
\l__nm_code_for_first_col_tl 149, 834
\l__nm_code_for_first_row_tl 153, 294
\l__nm_code_for_last_col_tl 151, 905
\l__nm_code_for_last_row_tl 155, 297
\g__nm_col_total_int 288, 289, 532, 701, 702, 890, 891, 1092, 2043, 2053, 2142
\c__nm_colortbl_loaded_bool . . . 63, 68, 475
\l__nm_columns_width_dim 52, 180, 235, 548, 673, 694, 2007, 2013
__nm_create_extra_nodes: 1124, 1291, 1292, 1784, 2033, 2120
__nm_create_nodes: 2080, 2118, 2138
\l__nm_ddots_int 1117, 1516, 1517
\g__nm_Ddots_lines_tl 566, 1138
__nm_define_com:nnn 2305, 2316, 2317, 2318, 2319, 2320
__nm_define_env:n 1003, 1022, 1023, 1024, 1025, 1026, 1027
\l__nm_delta_x_one_dim . . . 1119, 1519, 1529
\l__nm_delta_x_two_dim . . . 1121, 1561, 1571
\l__nm_delta_y_one_dim . . . 1120, 1521, 1529
\l__nm_delta_y_two_dim . . . 1122, 1563, 1571
__nm_dotfill: 1835, 1837, 1877
__nm_double_int_eval:n . . 1789, 1796, 1797
\g__nm_dp_ante_last_row_dim 303, 491, 492, 779
\g__nm_dp_last_row_dim 303, 304, 323, 324, 495, 496, 753, 779
\g__nm_dp_row_zero_dim 311, 312, 485, 486, 747
\c__nm_draft_bool 11, 12, 36, 414, 1731, 1802, 1834, 1886
__nm_draw_Cdots:nn 1400
__nm_draw_Ddots:nn 1495
__nm_draw_Hdotsfor:nnn 1741, 1749
__nm_draw_Iddots:nn 1537
__nm_draw_Ldots:nn 1349
__nm_draw_tikz_line: 1398, 1444, 1491, 1533, 1575, 1579, 1826, 1941
__nm_draw_Vdots:nn 1448
__nm_end_Cell: . 125, 326, 369, 505, 506, 507
\g__nm_env_int 50, 343, 397, 587, 600, 603, 681, 698, 711, 723, 862, 940, 1038, 1051, 1061, 1102, 1148, 1218, 1279, 1302, 1308, 1321, 1325, 1334, 1341, 1367, 1382, 1415, 1430, 1677, 1812, 1814, 1858, 1859, 1895, 1927, 2055, 2058, 2068, 2123, 2126, 2133, 2155, 2191
__nm_error:n . 18, 224, 228, 234, 243, 246, 255, 257, 265, 267, 273, 278, 739, 1083, 1884
__nm_error:nn 19, 186
__nm_error:nnn 20, 1829
__nm_everycr: 441, 480, 483
__nm_everycr_i: 442, 443
\l__nm_exterior_arraycolsep_bool 132, 231, 650, 659, 1867
\l__nm_extra_left_margin_dim 145, 170, 665, 878
\g__nm_extra_nodes_bool 140, 472, 550, 1124, 1857, 2212
\l__nm_extra_nodes_bool 139, 163, 472
\l__nm_extra_right_margin_dim 146, 171, 731, 926
__nm_extract_coords: 2168, 2175
__nm_fatal:n 21, 59, 579
__nm_fatal:nn 22
\l__nm_final_i_int 1127, 1168, 1173, 1176, 1197, 1202, 1208, 1219, 1226, 1309, 1383, 1431, 1754, 1776
\l__nm_final_j_int 1128, 1169, 1174, 1182, 1188, 1198, 1202, 1209, 1220, 1310, 1384, 1432, 1771, 1777, 1779
\l__nm_final_open_bool 1130, 1175, 1179, 1185, 1191, 1195, 1223, 1292, 1330, 1377, 1394, 1425, 1442, 1463, 1476, 1510, 1552, 1602, 1613, 1636, 1637, 1752, 1772, 1780, 1783, 1809, 1893
__nm_find_extremities_of_line:nnnn 1163, 1354, 1405, 1453, 1500, 1542
\l__nm_first_col_int 83, 84, 193, 271, 285, 645, 677, 756, 1869, 1947, 2043, 2053, 2082, 2142, 2287, 2293, 2299
\l__nm_first_row_int 81, 82, 194, 275, 530, 744, 763, 1951, 2036, 2050, 2081, 2140, 2285, 2569, 2578, 2587, 2596, 2605, 2622
__nm_gobble_ampersands:n 2206, 2225, 2230
__nm_Hdotsfor: 515, 526, 1726
__nm_Hdotsfor_i 1729, 1733, 1737
\g__nm_Hdotsfor_lines_tl . . 568, 1136, 1739
__nm_hdottedline: 513, 1853
\l__nm_hlines_bool 135, 158, 446
__nm_Hspace: 514, 1709
\g__nm_ht_last_row_dim 305, 321, 322, 493, 494, 753
\g__nm_ht_row_one_dim 318, 319, 489, 490, 768
\g__nm_ht_row_zero_dim 313, 314, 487, 488, 747, 768
__nm_i: 2036, 2038, 2039, 2040, 2041, 2050, 2055, 2059, 2060, 2061, 2062, 2068, 2069, 2070, 2071, 2083, 2085, 2088, 2089, 2093, 2094, 2140, 2146, 2149, 2155, 2158, 2170, 2179, 2184, 2191, 2194
\l__nm_iddots_int 1118, 1558, 1559
\g__nm_Iddots_lines_tl 567, 1139
__nm_if_not_empty_cell:nn 1028
__nm_if_not_empty_cell:nnTF 1202, 1262, 1762, 1776
\l__nm_impossible_line_bool 62, 1227, 1288, 1353, 1355, 1404, 1406, 1452, 1454, 1499, 1501, 1541, 1543
\l__nm_in_env_bool 54, 579, 580
\l__nm_initial_i_int 1125, 1166, 1233, 1236, 1257, 1263, 1269, 1280, 1287, 1303, 1368, 1416, 1753, 1762
\l__nm_initial_j_int 1126, 1167, 1234, 1242, 1248, 1258, 1263, 1270, 1281, 1304, 1369, 1417, 1757, 1763, 1765
\l__nm_initial_open_bool 1129, 1235, 1239, 1245, 1251, 1255, 1284, 1291, 1318, 1362, 1392, 1410, 1440, 1458, 1471, 1505, 1547, 1600, 1635, 1751, 1758, 1766, 1783, 1808, 1892
__nm_instruction_of_type:n 415, 417, 1681, 1687, 1693, 1699, 1705

\l__nm_inter_dots_dim 73, 74, 1134, 1605, 1609, 1616, 1620, 1626, 1631, 1643, 1651, 1840, 1843
__nm_j: 2043, 2045, 2046, 2047, 2048, 2053, 2055, 2059, 2062, 2064, 2065, 2068, 2071, 2073, 2074, 2096, 2098, 2102, 2104, 2108, 2109, 2142, 2145, 2148, 2155, 2158, 2171, 2178, 2183, 2191, 2194
\l__nm_l_dim 1581, 1582, 1598, 1605, 1609, 1616, 1620, 1626, 1631, 1643, 1644, 1651, 1652
\l__nm_last_col_bool 2595
\g__nm_last_col_found_bool 90, 562, 700, 809, 888, 1093
\l__nm_last_col_int 88, 89, 256, 266, 274, 654, 1013, 1015, 2289, 2295, 2301, 2528, 2536, 2544, 2552, 2560, 2568, 2577, 2586, 2604, 2613, 2621
\l__nm_last_row_int 85, 86, 195, 276, 296, 450, 595, 602, 609, 733, 737, 740, 750, 772, 829, 831, 900, 902, 1095, 1955, 1962, 1970, 1980, 1987, 2297, 2364
\l__nm_last_row_without_value_bool 87, 597, 735, 1097
\g__nm_last_vdotted_col_int 551, 553, 559, 561
\g__nm_Ldots_lines_tl 564, 1141
\g__nm_left_delim_dim . . . 615, 619, 632, 876
\l__nm_left_margin_dim 141, 164, 664, 877, 1875, 2113
\l__nm_letter_for_dotted_lines_str 242, 248, 249, 539, 540, 2515
__nm_line:nn 1151, 1791
__nm_line_i:nn 1795, 1800
\g__nm_max_cell_width_dim 330, 331, 589, 690, 2000, 2026
__nm_msg_new:nn 23, 34, 2331, 2337, 2344, 2350, 2356, 2362, 2370, 2375, 2381, 2386, 2391, 2501, 2506, 2513, 2519
__nm_msg_new:nnn 24, 2396, 2422, 2456, 2487
__nm_msg_redirect_name:nn 25, 237
__nm_multicolumn:nnn 516, 1715
\g__nm_multicolumn_cells_seq 528, 1720, 2062, 2071, 2164
\g__nm_multicolumn_sizes_seq 529, 1722, 2165
\l__nm_name_str 138, 188, 347, 349, 401, 403, 598, 607, 610, 866, 868, 944, 946, 1105, 1109, 2157, 2158, 2193, 2194
\g__nm_names_seq 53, 185, 187, 2499
\l__nm_nb_cols_int 2276, 2281, 2284, 2288, 2294, 2300
\l__nm_nb_rows_int 2275, 2280, 2291
\l__nm_NiceArray_bool 55, 591, 617, 648, 657, 761, 957, 1865
\g__nm_NiceMatrixBlock_int 51, 2006, 2011, 2014, 2025
__nm_node_for_multicolumn:nn . . 2166, 2173
\l__nm_nullify_dots_bool 136, 162, 1682, 1688, 1694, 1700, 1706
__nm_old_multicolumn 1714, 1717
\l__nm_parallelize_diags_bool 133, 134, 159, 1115, 1514, 1556
\l__nm_pos_env_str 130, 131, 261, 262, 263, 439, 765, 774
__nm_pre_array: 458, 614
\c__nm_preamble_first_col_tl 646, 816
\c__nm_preamble_last_col_tl 655, 884
\l__nm_radius_dim 75, 76, 1133, 1660
\l__nm_renew_dots_bool . 160, 230, 518, 2323
__nm_renew_matrix: 222, 225, 229, 2255, 2325
__nm_renew_NC@rewrite@S: 116, 560
__nm_renewcolumntype:nn 359, 536, 537
__nm_restore_iRow_jCol: . 1084, 1156, 1158
__nm_retrieve_coords:nn 1294, 1315, 1360, 1408, 1456, 1469, 1503, 1545
\c__nm_revtex_bool 27, 29, 32, 430
\g__nm_right_delim_dim . . 616, 620, 640, 924
\l__nm_right_margin_dim 142, 166, 730, 925, 1875, 2116
\g__nm_row_total_int 531, 1094, 1103, 1110, 2036, 2050, 2140
\l__nm_save_iRow_int 79, 461, 1160
\l__nm_save_jCol_int 80, 464, 1161
__nm_set_size:n 2273, 2282
\c__nm_siunitx_loaded_bool 91, 95, 100, 560
\l__nm_small_bool 157, 292, 467, 823, 894, 1131, 1839, 1846
\l__nm_stop_loop_bool 1170, 1171, 1199, 1203, 1230, 1231, 1259, 1264
\c__nm_table_collect_begin_tl 108, 110, 123
\c__nm_table_print_tl 111, 112, 125
__nm_test_if_math_mode: 56, 578, 967, 975, 983, 991, 999
\l__nm_the_array_box 643, 663, 784, 797
\g__nm_type_env_str 77, 572, 574, 958, 959, 965, 966, 973, 974, 981, 982, 989, 990, 997, 998, 1007, 1155, 2309, 2339, 2346, 2365, 2377, 2459, 2508, 2516
\g__nm_Vdots_lines_tl 565, 1137
__nm_vdottedline:n 555, 1881
__nm_vline: 590, 1945
__nm_vline_i: 69, 1956, 1963, 1971, 1981, 1988, 1993
\g__nm_width_first_col_dim 144, 759, 842, 845
\g__nm_width_last_col_dim 143, 811, 913, 916
\g__nm_x_final_dim 1298, 1311, 1344, 1468, 1481, 1487, 1489, 1520, 1528, 1562, 1570, 1588, 1625, 1641, 1806, 1823, 1890, 1901, 1923, 1935, 1938
\g__nm_x_initial_dim . . 1296, 1305, 1326, 1468, 1481, 1484, 1487, 1489, 1520, 1528, 1562, 1570, 1589, 1625, 1639, 1641, 1659, 1662, 1804, 1820, 1888, 1900, 1919, 1932, 1937
\g__nm_y_final_dim 1299, 1312, 1393, 1395, 1397, 1441, 1443, 1522, 1525, 1564, 1567, 1592, 1630, 1649, 1807, 1824, 1891, 1902, 1924
\g__nm_y_initial_dim 1297, 1306, 1393, 1395, 1396, 1441, 1443, 1522, 1527, 1564, 1569, 1593, 1630, 1647, 1649, 1659, 1663, 1805, 1821, 1889, 1904, 1910, 1911, 1920
\noalign 442, 479, 1855
\node . . 340, 394, 857, 935, 2150, 2186, 2244, 2251
\normalbaselines 466
\nulldelimiterspace 624, 635

O

\omit 677, 678, 705, 716

P

\path . . . 1818
peek commands:
\peek_charcode_remove_ignore_spaces:NTF . . . 2229
\pgfpathcircle . . . 1658
\pgfpoint . . . 1659
\pgfpointanchor . . . 1065, 1067
\pgfusepath . . . 1661
\phantom . . . 1682, 1688, 1694, 1700, 1706
\pNiceArray . . . 2529, 2570
\pNiceMatrix . . . 2258
prg commands:
\prg_do_nothing: . . . 104, 113, 479, 1835
\prg_replicate:nn . . . 701, 702, 1734, 1746, 2288, 2291, 2294, 2300
\prg_return_false: . . . 1042, 1055, 1072
\prg_return_true: . . . 1033, 1073
\prg_set_conditional:Npnn . . . 1028
\ProcessKeysOptions . . . 2330
\ProcessOptions . . . 14
\ProvideDocumentCommand . . . 38
\ProvidesExplPackage . . . 6

Q

quark commands:
\q_stop . . . 1789, 1796, 1797, 2168, 2175, 2199, 2200, 2273, 2282

R

\raise . . . 43, 45, 47
\relax . . . 14, 534, 535
\renewcommand . . . 118
\RenewDocumentEnvironment . . . 2257, 2260, 2263, 2266, 2269
\RequirePackage . . . 2, 4, 5, 15, 16, 17
\right . . . 628, 637, 802

S

\scriptstyle . . . 292, 823, 894, 1846
seq commands:
\seq_gclear_new:N . . . 528, 529
\seq_gput_left:Nn . . . 187, 1720, 1722
\seq_if_in:NnTF . . . 185, 2062, 2071
\seq_mapthread_function:NNN . . . 2163
\seq_new:N . . . 53
\seq_use:Nnnn . . . 2499
skip commands:
\skip_horizontal:N . . . 679, 696, 707, 718, 726
\skip_horizontal:n . . . 544, 664, 665, 682, 719, 730, 731, 758, 759, 796, 798, 811, 812, 874, 881, 919, 922, 1871
\skip_vertical:n . . . 453, 793, 800
\c_zero_skip . . . 484, 500
str commands:
\c_backslash_str . . . 2310
\c_colon_str . . . 249
\str_gclear:N . . . 1155
\str_gset:Nn . . . 574, 959, 966, 974, 982, 990, 998, 1007, 2309
\str_if_empty:NTF . . . 347, 401, 572, 598, 866, 944, 958, 965, 973, 981, 989, 997, 1105, 2157, 2193
\str_if_eq:nnTF . . . 178, 233, 765, 774
\str_new:N . . . 77, 130, 138, 248
\str_set:Nn . . . 131, 184, 242, 261, 262, 263
\str_set_eq:NN . . . 188, 249
\l_tmpa_str . . . 184, 185, 187, 188

T

\tabskip . . . 484, 500
TeX and LaTeX 2_ε commands:
\@acol . . . 434
\@acoll . . . 432
\@acolr . . . 433
\@addtopreamble . . . 590
\@array@array . . . 436
\@arrayacol . . . 432, 433, 434
\@arrayrule . . . 590
\@arstrutbox . . . 486, 488, 490, 492, 494, 496
\@currenvir . . . 2521
\@halignto . . . 435
\@height . . . 452
\@ifclassloaded . . . 28, 31
\@ifnextchar . . . 533
\@ifpackageloaded . . . 66, 94
\@mainaux . . . 1099, 1100, 1107, 1113, 2021, 2022, 2028, 2130, 2131, 2136
\@temptokena . . . 103, 106, 120, 122
\c@iRow . . . 293, 296, 302, 309, 316, 344, 350, 398, 404, 423, 448, 450, 461, 462, 530, 737, 740, 826, 831, 863, 869, 897, 902, 941, 947, 1080, 1094, 1096, 1160, 1176, 1676, 1721, 1742, 1922, 1934, 1953, 1955, 1960, 1962, 1968, 1970, 1978, 1980, 1985, 1987, 2083, 2216, 2218, 2238, 2365, 2366
\c@jCol . . . 282, 283, 289, 294, 345, 351, 399, 405, 424, 445, 464, 465, 551, 553, 555, 889, 891, 942, 948, 1092, 1093, 1161, 1188, 1248, 1335, 1342, 1676, 1721, 1724, 1743, 1769, 1905, 1949, 1976, 2096, 2115, 2127, 2217, 2219, 2239
\c@MaxMatrixCols . . . 1014
\CT@arc@ . . . 69
\CT@everycr . . . 477
\CT@row@color . . . 479
\ifmeasuring@ . . . 183
\NC@find . . . 104, 127
\NC@find@W . . . 535
\NC@find@w . . . 534
\NC@rewrite@S . . . 105, 118
\new@ifnextchar . . . 533
\p@ . . . 43, 45, 47
\pgf@x . . . 1066, 1068, 1305, 1311, 1326, 1344, 1820, 1823, 1900, 1901, 1919, 1923, 1932, 1935, 2065, 2074, 2124, 2128
\pgf@y . . . 1306, 1312, 1821, 1824, 1902, 1911, 1920, 1924, 2061, 2070
\pgfutil@firstofone . . . 1301, 1307, 1324, 1339, 1819, 1822, 1898, 1908, 1917, 1921, 1930, 1933, 2057, 2067, 2122, 2125
\tikz@library@external@loaded . . . 581, 1090
\tikz@parse@node . . . 1301, 1307, 1324, 1339, 1819, 1822, 1898, 1908, 1917, 1921, 1930, 1933, 2057, 2067, 2122, 2125
tex commands:
\tex_the:D . . . 122
\theiRow . . . 460, 1160

\thejCol 463, 1161
\tikz 333, 382, 387, 393, 680, 697, 708, 720, 850, 928
\tikzset 583, 585, 1091, 1142, 2079, 2117
tl commands:
\tl_const:Nn 816, 884
\tl_count:n 241
\tl_gclear:N 1153
\tl_gclear_new:N 563, 564, 565, 566, 567, 568
\tl_gput_left:Nn 2213
\tl_gput_right:Nn 419, 554, 1739
\tl_gset:Nn 106, 110, 112
\tl_if_empty:nTF 254, 264, 272
\tl_item:Nn 109, 110, 112
\tl_new:N 78, 108, 111
\tl_put_left:Nn 646, 651
\tl_put_right:Nn 655, 660
\tl_set:Nn 109, 644, 1058
\tl_set_rescan:Nnn 538
\tl_use:N 2393, 2398, 2424, 2458
\g_tmpa_tl 106, 109, 112
\l_tmpa_tl 109, 110, 644, 646, 651, 655, 660, 667, 1058, 1065, 1067
token commands:
\token_to_str:N 2333, 2378, 2399

U

\unless 183
use commands:
\use:N 422, 603, 610, 1049, 2014
\use:n 1793
\usetikzlibrary 3

V

\vbox 47
vbox commands:
\vbox:n 377
\vcenter 628, 637, 791, 2378
\Vdots 510
\vdots 522, 1670
Vdots internal commands:
__nm_Vdots 510, 522, 1691
vdots internal commands:
__nm_vdots 1670, 1694
\vline 69, 1993
\VNiceArray 2561, 2606
\vNiceArray 2553, 2597
\VNiceMatrix 2264
\vNiceMatrix 2261

Contents

1 Presentation 1

2 The environments of this extension 2

3 The continuous dotted lines 2
3.1 The option nullify-dots 3
3.2 The command \Hdotsfor 4
3.3 How to generate the continuous dotted lines transparently 5

4 The Tikz nodes created by nicematrix 5

5 The code-after 6

6 The environment {NiceArray} 7

7 The exterior rows and columns 7

8 The dotted lines to separate rows or columns 9

9 The width of the columns 10

10 Block matrices 11

11 The option small 11

12 The counters iRow and jCol 12

13 The option hlines 12

14 Utilisation of the column type S of siunitx 13

15 Technical remarks 13
15.1 Intersections of dotted lines . . . 13
15.2 Diagonal lines . . . 13
15.3 The “empty” cells . . . 14
15.4 The option exterior-arraycolsep . . . 15
15.5 The class option draft . . . 15
15.6 A technical problem with the argument of \\ . . . 15
15.7 Obsolete environments . . . 15

16 Examples 16
16.1 Dotted lines . . . 16
16.2 Width of the columns . . . 18
16.3 How to highlight cells of the matrix . . . 19
16.4 Direct utilisation of the Tikz nodes . . . 21

17 Implementation 22
17.1 Declaration of the package and extensions loaded . . . 23
17.2 Technical definitions . . . 23
17.2.1 Variables for the exterior rows and columns . . . 26
17.2.2 The column S of siunitx . . . 27
17.3 The options . . . 28
17.4 Important code used by {NiceArrayWithDelims} . . . 32
17.5 The environment {NiceArrayWithDelims} . . . 40
17.6 The environment {NiceMatrix} and its variants . . . 48
17.7 How to know whether a cell is “empty” . . . 49
17.8 After the construction of the array . . . 50
17.9 The actual instructions for drawing the dotted line with Tikz . . . 61
17.10User commands available in the new environments . . . 63
17.11The command \line accessible in code-after . . . 65
17.12The commands to draw dotted lines to separate columns and rows . . . 66
17.13The vertical rules . . . 69
17.14The environment {NiceMatrixBlock} . . . 70
17.15The extra nodes . . . 71
17.16Block matrices . . . 74
17.17How to draw the dotted lines transparently . . . 76
17.18Automatic arrays . . . 76
17.19We process the options . . . 77
17.20Error messages of the package . . . 77
17.21Obsolete environments . . . 81

18 History 82

Index 85