The package nicematrix[*]

F. Pantigny
fpantigny@wanadoo.fr

July 2, 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.

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

- continuous dotted lines[1];
- a first row and a last column for labels;
- a control of the width of the columns.

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

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 2.2.1 of nicematrix, at the date of 2019/07/02.

[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}`	`{pNiceArrayC}`	`{pNiceArrayRC}`
`{pNiceMatrix}`		`{bNiceArrayC}`	`{bNiceArrayRC}`
`{bNiceMatrix}`		`{BNiceArrayC}`	`{BNiceArrayRC}`
`{BNiceMatrix}`		`{vNiceArrayC}`	`{vNiceArrayRC}`
`{vNiceMatrix}`		`{VNiceArrayC}`	`{VNiceArrayRC}`
`{VNiceMatrix}`		`{NiceArrayCwithDelims}`	`{NiceArrayRCwithDelims}`

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{...}{...}`, `|`, `>{...}`, `<{...}`, `@{...}`, `!{...}` and `*{n}{...}` but the letters `p`, `m` and `b` should not be used. See p. 7 the section relating to `{NiceArray}`.

The environments with `C` at the end of their name, `{pNiceArrayC}`, `{bNiceArrayC}`, `{BNiceArrayC}`, `{vNiceArrayC}` and `{VNiceArrayC}` are similar to the environment `{NiceArray}` (especially the special letters `L`, `C` and `R`) but create an exterior column (on the right of the closing delimiter). See p. 8 the section relating to `{pNiceArrayC}`.

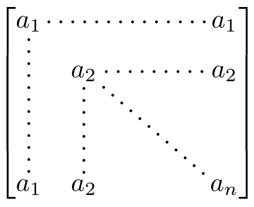
The environments with `RC`, `{pNiceArrayRC}`, `{bNiceArrayRC}`, `{BNiceArrayRC}`, `{vNiceArrayRC}`, `{VNiceArrayRC}` are similar to the environment `{NiceArray}` but create an exterior row (above the main matrix) and an exterior column. See p. 8 the section relating to `{pNiceArrayRC}`.

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`.[2]

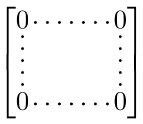
Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells[3] on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones.

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



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}
```



[2]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.

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

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

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

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

In the first column of this exemple, there are two instructions `\Vdots` but only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF[4]).
However, useless computations are performed by TeX before detecting that both instructions would eventually yield the same dotted line. That's why the package nicematrix provides starred versions of `\Ldots`, `\Cdots`, etc.: `\Ldots*`, `\Cdots*`, etc. These versions are simply equivalent to `\hphantom{\ldots}`, `\hphantom{\cdots}`, etc. The user should use these starred versions whenever a classical version has already been used for the same dotted line.

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

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

In fact, in this example, it would be possible to draw the same matrix without starred commands with the following code:

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

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

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.[5]
However, a command `\hspace*` might interfer with the construction of the dotted lines. That's why the package nicematrix provides a command `\Hspace` which is a variant of `\hspace` transparent for the dotted lines of nicematrix.

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

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

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

[5]Nevertheless, the best way to fix the width of a column is to use the environment `{NiceArray}` with a column of type `w` (or `W`).

3.1 The option nullify-dots

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

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

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

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

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

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

By default, with nicematrix, if we replace `{pmatrix}` by `{pNiceMatrix}` and `\vdots` by `\Vdots` (or `\Vdots*` for efficiency), 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.

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

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

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

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

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

The command `\hdotsfor` of amsmath takes an optional argument (between square brackets) which is used for fine tuning of the space beetween 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` is compatible with the extension colortbl.

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 `{matrix}`. This option can be set as option of `\usepackage` or with the command `\NiceMatrixOptions`.
In fact, this option is an alias for the conjonction of two options: `renew-dots` and `renew-matrix`.

- The option `renew-dots`

  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`[6] 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}
```

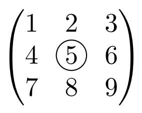
$$\begin{pmatrix} 1 & \cdots & \cdots & 1 \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

[6]The command `\iddots` is not a command of LaTeX but is defined by the package nicematrix. If `mathdots` is loaded, the version of `mathdots` is used.

4 The Tikz nodes created by nicematrix

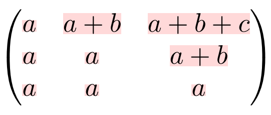
The package nicematrix creates a Tikz node for each cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix. However, the user may wish to use directly these nodes. It's possible. First, the user have to give a name to the array (with the key called `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) ;
```



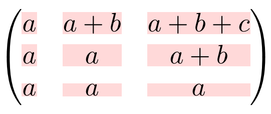
Don't forget the options `remember picture` and `overlay`.

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

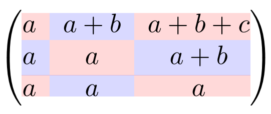


In fact, the package nicematrix can create "extra nodes". These new nodes are created if the option `create-extra-nodes` is used. There are two series of extra nodes: the "medium nodes" and the "large nodes".

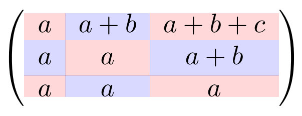
The names of the "medium nodes" are constructed by adding the suffix "`-medium`" to the names of the "normal nodes". In the following example, we have underlined the "medium nodes". We consider that this example is self-explanatory.



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.[7]



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`.[8]



[7]In the environments like `{pNiceArrayC}` and `{pNiceArrayRC}`, there is not "large nodes" created in the exterior row and column.

[8]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`.

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". In the following example, we have used `extra-left-margin` and `extra-right-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.

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.

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

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

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`[9] 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.[10]

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):

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

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

[9]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.
[10]In a command `\multicolumn`, one should also use the letters `L`, `C`, `R`.

An example where we use {NiceArray} because we want to use the types L and R for the columns:

```
$\left(\begin{NiceArray}{LCR}
a_{11}    & \Cdots & a_{1n} \\
a_{21}    &        & a_{2n} \\
\Vdots    &        & \Vdots \\
a_{n-1,1} & \Cdots & a_{n-1,n} \\
\end{NiceArray}\right)$
```

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & & a_{2n} \\ \vdots & & \vdots \\ a_{n-1,1} & \cdots & a_{n-1,n} \end{pmatrix}$$

7 The environment {pNiceArrayC} and its variants

The environment {pNiceArrayC} composes a matrix with an exterior column.
The environment {pNiceArrayC} takes a mandatory argument which is the preamble of the array. The types of columns available are the same as for the environment {NiceArray}. **However, no specification must be given for the last column.** It will automatically (and necessarily) be a L column.
A special option, called code-for-last-col, specifies tokens that will be inserted before each cell of the last column. The option columns-width doesn't apply to this external column.

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

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

In fact, the environment {pNiceArrayC} and its variants are based upon a more general environment, called {NiceArrayCwithDelims}. 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 {NiceArrayCwithDelims} if we want to use atypical delimiters.

```
$\begin{NiceArrayCwithDelims}
   {\downarrow}{\downarrow}{CCC}
1 & 2 & 3 & L_1 \\
4 & 5 & 6 & L_2 \\
7 & 8 & 9 & L_3
\end{NiceArrayCwithDelims}$
```

$$\left\downarrow\begin{array}{ccc}1&2&3\\4&5&6\\7&8&9\end{array}\right\downarrow\begin{array}{l}L_1\\L_2\\L_3\end{array}$$

8 The environment {pNiceArrayRC} and its variants

The environment {pNiceArrayRC} composes a matrix with an exterior row and an exterior column.
This environment {pNiceArrayRC} takes a mandatory argument which is the preamble of the array. As for the environment {pNiceArrayC}, no specification must be given for the last column (it will automatically be a L column).

A special option, called `code-for-first-row`, specifies tokens that will be inserted before each cell of the first row.

```
$\begin{pNiceArrayRC}{CCC}% (here, % is mandatory)
  [columns-width = auto,
   code-for-first-row = \color{blue},
   code-for-last-col  = \color{blue}]
C_1 & C_2 & C_3 \\
1 & 2 & 3 & L_1\\
4 & 5 & 6 & L_2\\
7 & 8 & 9 & L_3\\
\end{pNiceArrayRC}$
```

$$\begin{array}{cc} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} & \\ \left(\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}\right) & \begin{array}{c} L_1 \\ L_2 \\ L_3 \end{array} \end{array}$$

The first row of an environment `{pNiceArrayRC}` has the number 0, and not 1. This number is used for the names of the Tikz nodes (the names of these nodes are used, for example, by the command `\line` in `code-after`).
For technical reasons, it's not possible to use the option of the command `\\` after the first row (the placement of the delimiters would be wrong).

In fact, the environment `{pNiceArrayRC}` and its variants are based upon an more general environment, called `{NiceArrayRCwithDelims}`. 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 `{NiceArrayRCwithDelims}` if we want to use atypical delimiters.

```
$\begin{NiceArrayRCwithDelims}
   {\downarrow}{\downarrow}{CCC}[columns-width=auto]
C_1 & C_2 & C_3 \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{NiceArrayRCwithDelims}$
```

$$\begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \left\downarrow\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}\right\downarrow \end{array}$$

If we want to write a linear system, we can use the following code, with a preamble `CCC|C`:

```
$\begin{pNiceArrayRC}{CCC|C}
C_1 & \Cdots & C_n \\
a_{11} & \Cdots & a_{1n} & b_1 \\
\Vdots &        & \Vdots & \Vdots \\
a_{n1} & \Cdots & a_{nn} & b_n \\
\end{pNiceArrayRC}$
```

$$\begin{array}{c} \begin{array}{ccc|} C_1 & \cdots & C_n \end{array} \\ \left(\begin{array}{ccc|c} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_n \end{array}\right) \end{array}$$

The resultat may seem disappointing. It's possible to suppress the vertical rule in the first row with the command `\multicolumn` in order to "reconstruct" the cell.

```
$\begin{pNiceArrayRC}{CCC|C}
C_1 & \Cdots & \multicolumn{1}{C}{C_n} \\
a_{11} & \Cdots & a_{1n} & b_1 \\
\Vdots &        & \Vdots & \Vdots \\
a_{n1} & \Cdots & a_{nn} & b_n \\
\end{pNiceArrayRC}$
```

$$\begin{array}{c} \begin{array}{ccc} C_1 & \cdots & C_n \end{array} \\ \left(\begin{array}{ccc|c} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_n \end{array}\right) \end{array}$$

On the other side, we may remark that an horizontal line (with `\hline` or `\hdashline` of arydshln) doesn't extend in the "exterior column" of an environment like `{pNiceArrayC}` or `{pNiceArrayRC}`.

```
$\begin{pNiceArrayC}{CCC}
a_{11} & \Cdots & a_{1n} & L_1 \\
\Vdots &          & \Vdots & \Vdots \\
a_{n1} & \Cdots & a_{nn} & L_n \\
\hdashline
S_1    & \Cdots   & S_n \\
\end{pNiceArrayC}$
```

$$\left(\begin{array}{ccc} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \\ \hdashline S_1 & \cdots & S_n \end{array}\right) \begin{array}{c} L_1 \\ \vdots \\ L_n \\ \\ \end{array}$$

9 The dotted lines to separate rows or columns

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

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

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

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 "first row" and the "last column" of the environments for the environments with such features (*e.g.* `{pNiceArrayRC}`).

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

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

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)
```

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

10 The width of the columns

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

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

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

It's also possible to fix the width of all the columns of a matrix directly with the option `columns-width` (in all the environments of nicematrix).

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

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

Note that the space inserted between two columns (equal to 2 `\arraycolsep`) is not suppressed.

It's possible to give the 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. **Two or three compilations may be necessary.**

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

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

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`.[11]

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

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

11 Technical remarks

11.1 Diagonal lines

By default, all the diagonal lines[12] 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:

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

[12] 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:

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

11.2 The "empty" cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cells on both sides. However, a 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` and their starred versions 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.

11.3 The option exterior-arraycolsep

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea.[13]
The environment `{matrix}` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of amsmath prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep` and `{NiceArray}` does likewise.
However, the user can change this behaviour with the boolean option `exterior-arraycolsep` of the command `\NiceMatrixOptions`. With this option, `{NiceArray}` will insert the same horizontal spaces as the environment `{array}`.
This option is only for "compatibility" since the package nicematrix provides a more precise control with the options `left-margin`, `right-margin`, `extra-left-margin` and `extra-right-margin`.

11.4 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 `:`).[14]
That's why, when the class option `draft` is used, the dotted lines are not drawn, for a faster compilation.

11.5 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}
```

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

11.6 Compatibility with the extension dcolumn

If we want to make nicematrix compatible with dcolumn, it's necessary to patch the commands `\DC@endcentre` and `\DC@endright` as follow.

[13]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@{}}`.

[14]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.

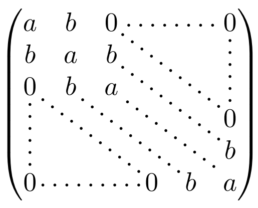
```
\def\DC@endcentre{$\egroup
\ifdim \wd\z@>\wd\tw@
\setbox\tw@=\hbox to\wd\z@{\unhbox\tw@\hfill}%
\else
\setbox\z@=\hbox to\wd\tw@{\hfill\unhbox\z@}\fi
\@@_Cell:\box\z@\box\tw@ \@@_end_Cell:}

\def\DC@endright{$\hfil\egroup \@@_Cell:\box\z@\box\tw@ \@@_end_Cell:}
```

12 Examples

12.1 Dotted lines

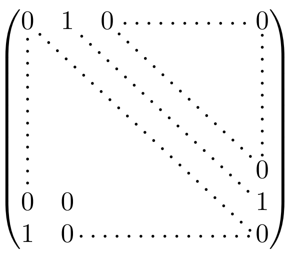
A tridiagonal matrix:

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



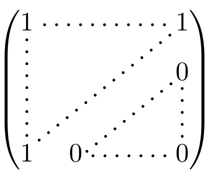
A permutation matrix:

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



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{pNiceMatrix}[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{pNiceMatrix}
```

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

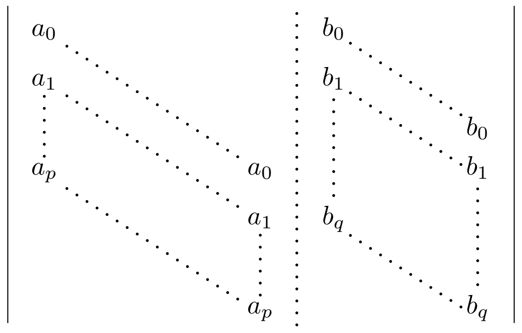
An example with `\Hdotsfor`:

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

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

An example for the resultant of two polynoms:

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



12.2 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 $\begin{pNiceArrayC}{CCCC:C}
1&1&1&1&1&\\
2&4&8&16&9&\\
3&9&27&81&36&\\
4&16&64&256&100&\\
\end{pNiceArrayC}$
...
\end{NiceMatrixBlock}
```

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

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

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

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

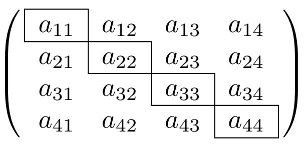
$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & -2 & -14 & -\frac{9}{2} \end{array}\right)\begin{array}{l} \\ \\ L_3 \gets \frac{1}{3}L_3 \\ \\ \end{array}$$

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

12.3 How to highlight cells of the matrix

In order to highlight a cell of a matrix, it's possible to "draw" one of the correspond 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`.

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



The package nicematrix is constructed upon the environment `{array}` and, therefore, it's possible to use the package colortbl in the environments of nicematrix.

```
$\begin{bNiceMatrix}
0 & \Cdots & 0 \\
\rowcolor{red!15} 1 & \Cdots & 1 \\
0 & \Cdots & 0 \\
\end{bNiceMatrix}$
```

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

The result may be disappointing. We therefore 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}}

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

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

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`.

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

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{array}\right)\begin{array}{l} L_1 \\ L_2 \\ L_3 \end{array}$$

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.

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{array}\right)\begin{array}{l} L_1 \\ L_2 \\ L_3 \end{array}$$

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.

$$\left(\begin{array}{ccc} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{array}\right)\begin{array}{l} L_1 \\ L_2 \\ L_3 \end{array}$$

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

```
\left(\,\begin{NiceArray}{>{\strut}CCCC}%
   [create-extra-nodes,left-margin,right-margin,
    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{NiceArray}\,\right)
```

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

12.4 Block matrices

In the following example, we use the "large nodes" to construct a block matrix (the dashed lines have been drawn with arydshln).

```
\NiceMatrixOptions{letter-for-dotted-lines = V}
\left(\begin{NiceArray}{CC:CC}%
   [create-extra-nodes,
    code-after = { \tikz \node [fit = (1-1-large) (2-2-large), inner sep = 0 pt]
                                   {$0_{22}$} ; } ]
       &        & a_{13} & a_{14} \\
       &        & a_{23} & a_{24} \\
\hdashline
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{34} & a_{44}
\end{NiceArray}\right)
```

$$D = \left(\begin{array}{cc:cc} \multicolumn{2}{c:}{\multirow{2}{*}{$0_{22}$}} & a_{13} & a_{14} \\ & & a_{23} & a_{24} \\ \hdashline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{34} & a_{44} \end{array}\right)$$

13 Implementation

By default, the package nicematrix doesn't patch any existing code.[15]

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 independant of its implementation. Unfortunately, it was not possible to be strictly independant: the package nicematrix relies upon the fact that the package `{array}` uses `\ialign` to begin the `\halign`.

The desire to do no modification to existing code leads to complications in the code of this extension.

13.1 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.[16]

```
\RequirePackage{tikz}
\usetikzlibrary{fit}
\RequirePackage{expl3}[2019/02/15]
```

[15]If we want nicematrix compatible with dcolumn, we have to patch dcolumn: cf. p. 13.

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

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_@@_draft_bool
\DeclareOption { draft } { \bool_set_true:N \c_@@_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 } [ 2018-10-17 ]
```

13.2 Technical definitions

We test whether the current class is revtex4-1 or revtex4-2.

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

```
\cs_new_protected:Nn \@@_error:n { \msg_error:nn { nicematrix } { #1 } }
```

```
\cs_new_protected:Nn \@@_error:nn { \msg_error:nn { nicematrix } { #1 } { #2 } }
```

```
\cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

```
\cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }
```

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

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

```
\ProvideDocumentCommand \iddots { }
  {
    \mathinner
      { \mkern 1 mu
        \raise \p@ \hbox { . }
        \mkern 2 mu
        \raise 4 \p@ \hbox { . }
        \mkern 2 mu
        \raise 7 \p@ \vbox { \kern 7 pt \hbox { . } } \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_@@_env_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width.

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

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

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

The integer `\l_@@_nb_first_row_int` is the number of the first row of the array. The default value is 1, but, in the environments like `{pNiceArrayRC}`, the value will be 0.

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

The flag `\l_@@_exterior_column_bool` will indicate if we are in an environment of the type of `{pNiceArrayC}` or `{pNiceArrayRC}`. It will be used for the creation of the "large nodes".

```
\bool_new:N \l_@@_exterior_column_bool
```

Consider the following code :

```
\begin{pNiceArrayC}{CC}
1 & 2
3 & 4
\end{pNiceArrayC}
```

In such a code, the last column of the environment `{pNiceArrayC}` is not used. We want to be able to detect such a situation. We create a boolean for that job.

```
\bool_new:N \g_@@_exterior_column_found_bool
```

This boolean will be raised in the last column of environments like `{pNiceArrayC}`.

We want to known if we are in an environment `{NiceArray}` because we want to raise an error if the user tries to use nested environments `{NiceArray}`.

```
\bool_new:N \l_@@_in_NiceArray_bool
```

A list of names of environments of the extension nicematrix which have a preamble as argument. This list is used to parameter some erros messages.

```
\seq_new:N \c_@@_env_with_preamble_seq
\seq_set_from_clist:Nn \c_@@_env_with_preamble_seq
  {
    NiceArray , NiceArrayCwithDelims , pNiceArrayC , vNiceArrayC , VNiceArrayC ,
    bNiceArrayC , BNiceArrayC , NiceArrayRCwithDelims , pNiceArrayRC ,
    vNiceArrayRC , VNiceArrayRC , bNiceArrayRC , BNiceArrayRC
  }
```

13.3 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}`, `{pNiceArrayC}`, `{pNiceArrayRC}` 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_@@_pos_env_str
\str_set:Nn \l_@@_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}` (but neither for `{NiceMatrix}`, `{pNiceArrayC}`, `{pNiceArrayRC}` and their variants even if these environments rely upon `{NiceArray}`).

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

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

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

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

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

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cell of the "exterior column" of an environment of the kind of `{pNiceArrayC}`).

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

The token list `\l_@@_code_for_last_col_tl` will contain code inserted at the beginning of each cell of the last column in the environment `{pNiceArrayC}` (and its variants). It corresponds to the option `code-for-last-col`.

```
\tl_new:N \l_@@_code_for_last_col_tl
```

We don't want to patch any existing code. That's why some code must be executed in a `\group_insert_after:N`. That's why the parameters used in that code must be transfered outside the current group. To do this, we copy those quantities in global variables just before the `\group_insert_after:N`. Therefore, for those quantities, we have two parameters, one local and one global. For example, we have `\l_@@_name_str` and `\g_@@_name_str`.

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 \g_@@_name_str
\str_new:N \l_@@_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_@@_extra_nodes_bool
\bool_new:N \g_@@_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_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim
\dim_new:N \g_@@_left_margin_dim
\dim_new:N \g_@@_right_margin_dim
```

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

```
\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim
\dim_new:N \g_@@_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 keys of set).

```
\keys_define:nn { NiceMatrix / Global }
  {
    parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
    parallelize-diags .default:n = true ,
```

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_@@_renew_dots_bool ,
    renew-dots .default:n = true ,
    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
    nullify-dots .default:n = true ,
```

An option to test whether the extra nodes will be created (these nodes are the “medium nodes” and “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_@@_extra_nodes_bool ,
    create-extra-nodes .default:n = true,
    left-margin .dim_set:N = \l_@@_left_margin_dim ,
    left-margin .default:n = \arraycolsep ,
    right-margin .dim_set:N = \l_@@_right_margin_dim ,
    right-margin .default:n = \arraycolsep ,
    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim
  }
```

We define a set of keys used by the environments (and 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_@@_auto_columns_width_bool }
        { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
    columns-width .value_required:n = true ,
    name .code:n =
      \seq_if_in:NnTF \g_@@_names_seq { #1 }
         { \@@_error:nn { Duplicate~name } { #1 } }
         { \seq_gput_left:Nn \g_@@_names_seq { #1 } }
      \str_set:Nn \l_@@_name_str { #1 } ,
    name .value_required:n = true ,
    code-after .tl_gset:N = \g_@@_code_after_tl ,
    code-after .initial:n = \c_empty_tl ,
    code-after .value_required:n = true ,
  }
```

We begin the construction of the major sets of keys (used by the differents 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
      } ,
    NiceArrayC .inherit:n =
      {
        NiceMatrix / Global ,
        NiceMatrix / Env
       } ,
    NiceArrayRC .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 = \@@_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    RenewMatrix .meta:n = renew-matrix ,
    transparent .meta:n = { renew-dots , renew-matrix } ,
    transparent .value_forbidden:n = true,
    Transparent .meta:n = transparent,
```

The following option is only for the environment `{pNiceArrayC}` and its variants. It will contain code inserted at the beginning of each cell of the last column.[17]

```
    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
```

Idem for the first row in environments like `{pNiceArrayRC}`.

```
    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
```

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

```
    exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
    exterior-arraycolsep .default:n  = true ,
```

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 =
```

[17]In an environment `{pNiceArrayC}`, the last column is composed outside the parentheses of the array.

```
    \str_if_eq:nnTF { #1 } { auto }
      { \@@_error:n { Option~auto~for~columns-width } }
      { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

Usually, an error is raised when the user tries to give the same 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 =
    \@@_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 `{pNiceArrayC}` 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 .tl_set:N = \l_@@_letter_for_dotted_lines_str ,
  letter-for-dotted-lines .value_required:n = true ,
  letter-for-dotted-lines .initial:n = \c_colon_str ,
```

```
  unknown .code:n  = \@@_error:n { Unknown~key~for~NiceMatrixOptions } }
```

```
\@@_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~keys~are~(in~alphabetic~order):~
    allow-duplicate-names,~
    code-for-last-col,~
    exterior-arraycolsep,~
    left-margin,~
    letter-for-dotted-lines,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots,~
    renew-matrix,~
    right-margin,~
    and~transparent
  }
\@@_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.
  }
```

`\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 }
  { unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix } }
```

```
\@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
  {
    The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
    \{NiceMatrix\}~and~its~variants. \\
    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,~
    columns-width,~
    create-extra-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots~
    and~right-margin.
  }
```

```
\@@_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'. \\
    For~a~list~of~the~names~already~used,~type~H~<return>. \\
    If~you~don't~want~to~see~this~message~again,~use~the~option~
    'allow-duplicate-names'.
  }
  {
    The~names~already~defined~in~this~document~are:~
    \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
  }
```

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_@@_pos_env_str c ,
    t .code:n = \str_set:Nn \l_@@_pos_env_str t ,
    b .code:n = \str_set:Nn \l_@@_pos_env_str b ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceArray }
  }
\@@_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,~
    create-extra-nodes,~
    columns-width,~
    extra-left-margin,~
    extra-right-margin,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots,~
    right-margin,~
    and~t.
  }
```

13.4 The environments {NiceArray} and {NiceMatrix}

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 \@@_Cell:
  {
```

We increment `\g_@@_column_int`, which is the counter of the columns.

```
    \int_gincr:N \g_@@_column_int
```

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 \g_@@_column_int = 1 { \int_gincr:N \g_@@_row_int }
    \int_gset:Nn \g_@@_column_total_int
       { \int_max:nn \g_@@_column_total_int \g_@@_column_int }
    \hbox_set:Nw \l_tmpa_box $ % $
    \int_compare:nNnT \g_@@_row_int = \c_zero_int
      \l_@@_code_for_first_row_tl
  }
\cs_new_protected:Nn \@@_end_Cell:
  { $ % $
    \hbox_set_end:
```

We want to compute in `\l_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the last column of an environment of the kind of `{pNiceArrayC}`).

```
    \dim_gset:Nn \g_@@_max_cell_width_dim
      { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_tmpa_box } }
    \int_compare:nNnT \g_@@_row_int = \c_zero_int
      {
        \dim_gset:Nn \g_@@_max_dp_row_zero_dim
          { \dim_max:nn \g_@@_max_dp_row_zero_dim { \box_dp:N \l_tmpa_box } }
        \dim_gset:Nn \g_@@_max_ht_row_zero_dim
          { \dim_max:nn \g_@@_max_ht_row_zero_dim { \box_ht:N \l_tmpa_box } }
      }
    \int_compare:nNnT \g_@@_row_int = \c_one_int
      {
        \dim_gset:Nn \g_@@_max_ht_row_one_dim
          { \dim_max:nn \g_@@_max_ht_row_one_dim { \box_ht:N \l_tmpa_box } }
      }
```

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_@@_env_int -
                    \int_use:N \g_@@_row_int -
                    \int_use:N \g_@@_column_int ,
        alias =
          \str_if_empty:NF \l_@@_name_str
            {
              \l_@@_name_str -
              \int_use:N \g_@@_row_int -
              \int_use:N \g_@@_column_int
            }
      ]
    \bgroup
    \box_use:N \l_tmpa_box
    \egroup ;
  }
```

The environment {NiceArray} is the main environment of the extension nicematrix. In order to clarify the explanations, we will first give the definition of the environment {NiceMatrix}.
Our environment {NiceMatrix} must have the same second part as the environment {matrix} of amsmath (because of the programmation of the option renew-matrix). Hence, this second part is the following:

```
\endarray
\skip_horizontal:n {-\arraycolsep}
```

That's why, in the definition of {NiceMatrix}, we must use \NiceArray and not \begin{NiceArray} (and, in the definition of {NiceArray}, we will have to use \array, and not \begin{array}: see below).

Here's the definition of {NiceMatrix}:

```
\NewDocumentEnvironment { NiceMatrix } { ! O { } }
  {
    \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
    \str_set:Nn \l_@@_pos_env_str c
    \bool_set_false:N \l_@@_exterior_arraycolsep_bool
    \NiceArray { * \c@MaxMatrixCols C }
  }
  {
    \endarray
    \skip_horizontal:n
      { \g_@@_right_margin_dim + \g_@@_extra_right_margin_dim - \arraycolsep }
  }
```

For the definition of {NiceArray} (just below), we have the following constraints:

- we must use \array in the first part of {NiceArray} and, therefore, \endarray in the second part;
- we have to put a \group_insert_after:N \@@_after_array: in the first part of {NiceArray} so that \@@_draw_lines will be executed at the end of the current environment (either {NiceArray} or {NiceMatrix}).

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

```
\@@_msg_new:nn { Yet~in~NiceArray }
  {
    Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~
    nested.~You~can~go~on,~but,~maybe,~you~will~have~errors~or~an~incorrect~
    result.
  }
```

The command `\@@_define_dots:` will be used in the environment `{NiceArray}` to define the commands `\Ldots`, `\Cdots`, etc.

```
\cs_new_protected:Npn \@@_define_dots:
  {
    \cs_set_eq:NN \Ldots \@@_Ldots
    \cs_set_eq:NN \Cdots \@@_Cdots
    \cs_set_eq:NN \Vdots \@@_Vdots
    \cs_set_eq:NN \Ddots \@@_Ddots
    \cs_set_eq:NN \Iddots \@@_Iddots
    \bool_if:NT \l_@@_renew_dots_bool
      {
        \cs_set_eq:NN \ldots \@@_Ldots
        \cs_set_eq:NN \cdots \@@_Cdots
        \cs_set_eq:NN \vdots \@@_Vdots
        \cs_set_eq:NN \ddots \@@_Ddots
        \cs_set_eq:NN \iddots \@@_Iddots
        \cs_set_eq:NN \dots \@@_Ldots
        \cs_set_eq:NN \hdotsfor \@@_Hdotsfor
      }
  }
```

With `\@@_define_dots_to_nil:`, the commands like `\Ldots`, `\Cdots`, are defined, but with no effect. This command will be used if the class option `draft` is used.

```
\cs_new_protected:Npn \@@_define_dots_to_nil:
  {
    \cs_set_eq:NN \Ldots \prg_do_nothing:
    \cs_set_eq:NN \Cdots \prg_do_nothing:
    \cs_set_eq:NN \Vdots \prg_do_nothing:
    \cs_set_eq:NN \Ddots \prg_do_nothing:
    \cs_set_eq:NN \Iddots \prg_do_nothing:
    \bool_if:NT \l_@@_renew_dots_bool
      {
        \cs_set_eq:NN \ldots \prg_do_nothing:
        \cs_set_eq:NN \cdots \prg_do_nothing:
        \cs_set_eq:NN \vdots \prg_do_nothing:
        \cs_set_eq:NN \ddots \prg_do_nothing:
        \cs_set_eq:NN \iddots \prg_do_nothing:
        \cs_set_eq:NN \dots \prg_do_nothing:
        \cs_set_eq:NN \hdotsfor \@@_Hdotsfor
      }
  }
```

In the environment `{NiceArray}`, 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 \@@_renewcolumntype:nn
  {
    \newcolumntype #1 [ 2 ]
      {
        > {
            \hbox_set:Nw \l_tmpa_box
            \@@_Cell:
          }
        c
        < {
            \@@_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 (@@~north~east) ;
                      }
                    \hbox:n
                      {
                        \tikz [ remember~picture , overlay ]
                        \coordinate (@@~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 contruct by a standard `node` instruction because such a construction give a 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_@@_env_int -
                            \int_use:N \g_@@_row_int -
                            \int_use:N \g_@@_column_int - w,
                alias =
                  \str_if_empty:NF \l_@@_name_str
                    {
                      \l_@@_name_str -
                      \int_use:N \g_@@_row_int -
                      \int_use:N \g_@@_column_int - w
                    } ,
                inner~sep = \c_zero_dim ,
                fit = (@@~south~west) (@@~north~east)
              ]
            ;
          }
      }
  }
```
First, we test if we are yet in an environment `{NiceArray}` (nested environments are forbidden).
```
\NewDocumentEnvironment { NiceArray } { O { } m ! O { } }
  {
    \bool_if:NT \l_@@_in_NiceArray_bool
      { \@@_error:n { Yet~in~NiceArray } }
```
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 } }
    \bool_set_true:N \l_@@_in_NiceArray_bool
    \group_insert_after:N \@@_after_array:
    \tl_gclear_new:N \g_@@_lines_to_draw_tl
```

We increment the counter `\g_@@_env_int` which counts the environments `{NiceArray}`.

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

For the following variables, maybe we should create it only if we use the environment `{pNiceArrayRC}` or its variants.

```
    \dim_gzero_new:N \g_@@_max_dp_row_zero_dim
    \dim_gzero_new:N \g_@@_max_ht_row_zero_dim
    \dim_gzero_new:N \g_@@_max_ht_row_one_dim

    \keys_set:nn { NiceMatrix / NiceArray } { #1 , #3 }
```

If the user requires all the columns to have a width equal to the widest cell of the array, we read this length in the file `.aux` (of, course, this is possible only on the second run of LaTeX : on the first run, the dimension `\l_@@_columns_width_dim` will be set to zero — and the columns will have their natural width).

```
    \bool_if:NT \l_@@_auto_columns_width_bool
      {
        \group_insert_after:N \@@_write_max_cell_width:
        \cs_if_free:cTF { _@@_max_cell_width_ \int_use:N \g_@@_env_int }
          { \dim_set:Nn \l_@@_columns_width_dim \c_zero_dim }
          {
            \dim_set:Nx \l_@@_columns_width_dim
              { \use:c { _@@_max_cell_width _ \int_use:N \g_@@_env_int } }
          }
```

If the environment has a name, we read the value of the maximal value of the columns from `_@@_name_cell_width`*`name`* (the value will be the correct value even if the number of the environment has changed (for example because the user has created or deleted an environment before the current one)).

```
        \str_if_empty:NF \l_@@_name_str
          {
            \cs_if_free:cF { _@@_max_cell_width_ \l_@@_name_str }
              {
                \dim_set:Nx \l_@@_columns_width_dim
                  { \use:c { _@@_max_cell_width_ \l_@@_name_str } }
              }
          }
      }
```

We don't want to patch any code and that's why some code is executed in a `\group_insert_after:N`. In particular, in this `\group_insert_after:N`, we will have to know the value of some parameters like `\l_@@_extra_nodes_bool`. That's why we transit via a global version for some variables.

```
    \bool_gset_eq:NN \g_@@_extra_nodes_bool \l_@@_extra_nodes_bool
    \dim_gset_eq:NN \g_@@_left_margin_dim \l_@@_left_margin_dim
    \dim_gset_eq:NN \g_@@_right_margin_dim \l_@@_right_margin_dim
    \dim_gset_eq:NN \g_@@_extra_right_margin_dim \l_@@_extra_right_margin_dim
    \tl_gset_eq:NN \g_@@_name_str \l_@@_name_str
```

The environment `{array}` uses internally the command `\ialign` and, in particular, this command `\ialign` sets `\everycr` to `{}`. However, we want to use `\everycr` in our array. The solution is to give to `\ialign` a new definition (giving to `\everycr` the value we want) that will revert automatically to its default definition after the first utilisation.[18]

```
    \cs_set:Npn \ialign
      {
        \everycr { \noalign { \int_gzero:N \g_@@_column_int } }
        \tabskip = \c_zero_skip
```

[18]With this programmation, we will have, in the cells of the array, a clean version of `\ialign`. That's necessary: the user will probably not employ directly `\ialign` in the array... but more likely environments that utilize `\ialign` internally (e.g.: `{substack}`)

```
        \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}`.

```
    \dim_compare:nNnTF \l_@@_columns_width_dim = \c_zero_dim
      {
       \newcolumntype L { > \@@_Cell: l < \@@_end_Cell: }
       \newcolumntype C { > \@@_Cell: c < \@@_end_Cell: }
       \newcolumntype R { > \@@_Cell: r < \@@_end_Cell: }
      }
```

If there is an option that specify that all the columns must have the same width, the column types `L`, `C` and `R` are in fact defined upon the column type `w` of array which is, in fact, redefined below.

```
      {
       \newcolumntype L { w l { \dim_use:N \l_@@_columns_width_dim } }
       \newcolumntype C { w c { \dim_use:N \l_@@_columns_width_dim } }
       \newcolumntype R { w r { \dim_use:N \l_@@_columns_width_dim } }
      }
```

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
```

We redefine the column types `w` and `W` of the package array.

```
    \@@_renewcolumntype:nn w { }
    \@@_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`.

```
    \exp_args:Nx \newcolumntype \l_@@_letter_for_dotted_lines_str
      {
        !
          {
            \skip_horizontal:n { 0.53 pt }
            \bool_gset_true:N \g_@@_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 \g_@@_column_int > \g_@@_last_vdotted_col_int
              {
                \int_gset_eq:NN \g_@@_last_vdotted_col_int \g_@@_column_int
                \tl_gput_right:Nx \g_@@_code_after_tl
                  {
                    \exp_not:N \@@_vdottedline:n
                    \exp_not:N {
                    \int_use:N \g_@@_column_int
                    \exp_not:N }
                  }
              }
          }
      }
```

The commands `\Ldots`, `\Cdots`, etc. will be defined only in the environment `{NiceArray}`. If the class option `draft` is used, these commands will be defined to be no-op (the dotted lines are not drawn).

```
    \bool_if:NTF \c_@@_draft_bool
      \@@_define_dots_to_nil:
      \@@_define_dots:
    \cs_set_eq:NN \hdottedline \@@_hdottedline:
    \cs_set_eq:NN \Hspace \@@_Hspace:
    \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor
    \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
```

The sequence `\g_@@_empty_cells_seq` will contain a list of "empty" cells (not all the empty cells of the matrix). If we want to indicate that the cell in row i and column j must be considered as empty, the token list "`i-j`" will be put in this sequence.

```
    \seq_gclear_new:N  \g_@@_empty_cells_seq
```

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

```
    \seq_gclear_new:N \g_@@_multicolumn_cells_seq
    \seq_gclear_new:N \g_@@_multicolumn_sizes_seq
```

The counter `\g_@@_row_int` will be used to count the rows of the array (its incrementation will be in the first cell of the row). At the end of the environment `{array}`, this counter will give the total number of rows of the matrix.

```
    \int_gzero_new:N \g_@@_row_int
    \int_gset:Nn \g_@@_row_int { \l_@@_nb_first_row_int - 1 }
```

The counter `\g_@@_column_int` 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_@@_column_total_int`. These counters are updated in the command `\@@_Cell:` executed at the beginning of each cell.

```
    \int_gzero_new:N \g_@@_column_int
    \int_gzero_new:N \g_@@_column_total_int

    \int_gzero_new:N \g_@@_last_vdotted_col_int
    \int_gset:Nn \g_@@_last_vdotted_col_int { -1 }
    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

The extra horizontal spaces on both sides of an environment `{array}` should be considered as a bad idea of standard LaTeX. In the environment `{matrix}` the package amsmath prefers to suppress these spaces with instructions "`\hskip -\arraycolsep`". In the same way, we decide to suppress them in `{NiceArray}`. However, for better compatibility, we give an option `exterior-arraycolsep` to control this feature.

```
    \bool_if:NF \l_@@_exterior_arraycolsep_bool
      { \skip_horizontal:n { - \arraycolsep } }

    \skip_horizontal:n { \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
```

Eventually, the environment `{NiceArray}` is defined upon the environment `{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`.

```
    \bool_if:NTF \c_@@_revtex_bool
```

```
        {
          \cs_set_eq:NN \@acoll \@arrayacol
          \cs_set_eq:NN \@acolr \@arrayacol
          \cs_set_eq:NN \@acol \@arrayacol
          \cs_set:Npn \@halignto { }
          \@array@array
        }
        \array
```

The token list `\l_@@_pos_env_str`, which is the argument of `\array` (or `\@array@array`) will contain one of the values `t`, `c` or `b`.

```
     [ \l_@@_pos_env_str ] { #2 }
  }
```

```
  { \endarray
     \bool_if:NF \l_@@_exterior_arraycolsep_bool
       { \skip_horizontal:n { - \arraycolsep } }
     \skip_horizontal:n
       { \g_@@_right_margin_dim + \g_@@_extra_right_margin_dim }
  }
```

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

```
\NewDocumentEnvironment { pNiceMatrix } { }
   { \left( \begin{NiceMatrix} }
   { \end{NiceMatrix} \right) }
\NewDocumentEnvironment { bNiceMatrix } { }
   { \left[ \begin{NiceMatrix} }
   { \end{NiceMatrix} \right] }
\NewDocumentEnvironment { BNiceMatrix } { }
   { \left\{ \begin{NiceMatrix} }
   { \end{NiceMatrix} \right\} }
\NewDocumentEnvironment { vNiceMatrix } { }
   { \left\lvert \begin{NiceMatrix} }
   { \end{NiceMatrix} \right\rvert }
\NewDocumentEnvironment { VNiceMatrix } {}
   { \left\lVert \begin{NiceMatrix} }
   { \end{NiceMatrix} \right\rVert }
```

For the option `columns-width=auto` (or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`), we want to know the maximal width of the cells of the array (except the cells of the "exterior" column of an environment of the kind of `{pNiceAccayC}`). This length can be known only after the end of the construction of the array (or at the end of the environment `{NiceMatrixBlock}`). That's why we store this value in the main `.aux` file and it will be available in the next run. We write a dedicated command for this because it will be called in a `\group_insert_after:N`.

```
\cs_new_protected:Nn \@@_write_max_cell_width:
  {
    \bool_if:NF \l_@@_block_auto_columns_width_bool
      {
        \iow_now:Nn \@mainaux \ExplSyntaxOn
        \iow_now:Nx \@mainaux
          {
            \cs_gset:cpn { @@_max_cell_width_ \int_use:N \g_@@_env_int }
              { \dim_use:N \g_@@_max_cell_width_dim }
          }
```

If the environment has a name, we also create an alias named `\@@_max_cell_width_`*`name`*.

```
    \iow_now:Nx \@mainaux
     {
      \cs_gset:cpn { @@_max_cell_width_ \g_@@_name_str }
       { \dim_use:N \g_@@_max_cell_width_dim }
     }
    \iow_now:Nn \@mainaux \ExplSyntaxOff
   }
 }
```

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 \@@_if_not_empty_cell:nn #1 #2 { T , TF }
```

If the cell is an implicit cell (that is after the symbol `\\` of end of row), the cell must, of course, be considered as empty. It's easy to check whether we are in this situation considering the correspondant Tikz node.

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

We manage a list of "empty cells" called `\g_@@_empty_cells_seq`. In fact, this list is not a list of all the empty cells of the array but only those explicitely declared empty for some reason. It's easy to check if the current cell is in this list.

```
   {
    \seq_if_in:NxTF \g_@@_empty_cells_seq { \int_use:N #1 - \int_use:N #2 }
     \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_@@_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:nTF
       { \dim_abs:n { \g_tmpb_dim - \g_tmpa_dim } < 0.5 pt }
       \prg_return_false:
       \prg_return_true:
     }
   }
 }
```

The argument of the following command `\@@_instruction_of_type:n` is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). This command writes in `\g_@@_lines_to_draw_tl` the instruction that 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 & \Hdotsfor{2} \\
\end{pNiceMatrix}
```

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

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

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

```
\cs_new_protected:Nn \@@_instruction_of_type:n
  {
    \tl_gput_right:Nx \g_@@_lines_to_draw_tl
      {
        \exp_not:c { @@ _ draw _ #1 : nn }
          { \int_use:N \g_@@_row_int }
          { \int_use:N \g_@@_column_int }
      }
  }
```

13.5 After the construction of the array

```
\cs_new_protected:Nn \@@_after_array:
  {
    \int_compare:nNnTF \g_@@_row_int > 0
      \@@_after_array_i:
      { \@@_error:n { Zero~row } }
  }
```

```
\@@_msg_new:nn { Zero~row }
  {
    There~is~a~problem.~Maybe~your~environment~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.
  }
```

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

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

Now, the definition of the counters `\g_@@_column_int` and `\g_@@_column_total_int` change: `\g_@@_column_int` will be the number of columns without the exterior column (in an environment like `{pNiceArrayC}`) and `\g_@@_column_total_int` will be the number of columns with this exterior column.

```
    \int_gset_eq:NN \g_@@_column_int \g_@@_column_total_int
    \bool_if:nT { \l_@@_exterior_column_bool && \g_@@_exterior_column_found_bool }
       { \int_gdecr:N \g_@@_column_int }
```

The sequence `\g_@@_yet_drawn_seq` contains a list of lines which have been drawn previously in the matrix. We maintain this sequence because we don't want to draw two overlapping lines.

```
    \seq_gclear_new:N \g_@@_yet_drawn_seq
```

By default, the diagonal lines will be parallelized[19]. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_zero_new:N \l_@@_ddots_int
        \int_zero_new:N \l_@@_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_@@_delta_x_one_dim
        \dim_zero_new:N \l_@@_delta_y_one_dim
        \dim_zero_new:N \l_@@_delta_x_two_dim
```

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

```
        \dim_zero_new:N \l_@@_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_@@_extra_nodes_bool \@@_create_extra_nodes:
```

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

```
    \tl_if_empty:NF \g_@@_lines_to_draw_tl
      {
        \int_zero_new:N \l_@@_initial_i_int
        \int_zero_new:N \l_@@_initial_j_int
        \int_zero_new:N \l_@@_final_i_int
        \int_zero_new:N \l_@@_final_j_int
        \bool_set_false:N \l_@@_initial_open_bool
        \bool_set_false:N \l_@@_final_open_bool
        \g_@@_lines_to_draw_tl
      }
    \tl_gclear:N \g_@@_lines_to_draw_tl
```

Now, the `code-after`.

```
    \tikzset
      {
        every~picture / .style =
         {
           overlay ,
           remember~picture ,
           name~prefix = nm - \int_use:N \g_@@_env_int -
         }
      }
    \cs_set_eq:NN \line \@@_line:nn
    \g_@@_code_after_tl
    \tl_gclear:N \g_@@_code_after_tl
    \group_end:
  }
```

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” (the medium and large nodes are created with `\@@_create_extra_nodes:` if they have not been created yet).

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

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 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 \@@_find_extremities_of_line:nnnn
 {
   \int_set:Nn \l_@@_initial_i_int { #1 }
   \int_set:Nn \l_@@_initial_j_int { #2 }
   \int_set:Nn \l_@@_final_i_int { #1 }
   \int_set:Nn \l_@@_final_j_int { #2 }
   \bool_set_false:N \l_@@_initial_open_bool
   \bool_set_false:N \l_@@_final_open_bool
```

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_@@_stop_loop_bool
   \bool_do_until:Nn \l_@@_stop_loop_bool
     {
       \int_add:Nn \l_@@_final_i_int { #3 }
       \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
        \bool_if:nTF
          {
              \int_compare_p:nNn
                \l_@@_final_i_int < { \l_@@_nb_first_row_int - 1 }
            || \int_compare_p:nNn \l_@@_final_i_int > \g_@@_row_int
            || \int_compare_p:nNn \l_@@_final_j_int < \c_one_int
            || \int_compare_p:nNn \l_@@_final_j_int > \g_@@_column_total_int
```

If you arrive in the column `C` of an environment with such columns (like `{pNiceArrayC}`), you must consider that we are *outside* the matrix except if we are drawing a vertical line (included in the column `C`).

```
            || \int_compare_p:nNn \l_@@_final_j_int > \g_@@_column_int
               && \int_compare_p:nNn { #4 } > \c_zero_int
          }
```

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

```
          {
            \bool_set_true:N \l_@@_final_open_bool
```

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

If we are in the matrix, we test if 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`.

```
          {
            \@@_if_not_empty_cell:nnT \l_@@_final_i_int \l_@@_final_j_int
              { \bool_set_true:N \l_@@_stop_loop_bool }
          }
      }
```

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

```
   \bool_set_false:N \l_@@_stop_loop_bool
```

```
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
        \int_sub:Nn \l_@@_initial_i_int { #3 }
        \int_sub:Nn \l_@@_initial_j_int { #4 }
        \bool_if:nTF
          {
            \int_compare_p:nNn \l_@@_initial_i_int < \l_@@_nb_first_row_int
              ||
            \int_compare_p:nNn \l_@@_initial_i_int > \g_@@_row_int
              ||
            \int_compare_p:nNn \l_@@_initial_j_int < 1
              ||
            \int_compare_p:nNn \l_@@_initial_j_int > \g_@@_column_total_int
          }
          {
            \bool_set_true:N \l_@@_initial_open_bool
            \int_add:Nn \l_@@_initial_i_int { #3 }
            \int_add:Nn \l_@@_initial_j_int { #4 }
            \bool_set_true:N \l_@@_stop_loop_bool
          }
          {
            \@@_if_not_empty_cell:nnT
              \l_@@_initial_i_int \l_@@_initial_j_int
              { \bool_set_true:N \l_@@_stop_loop_bool }
          }
      }
```

If we have at least one open extremity, we create the "medium nodes" in the matrix (in the case of an open extremity, the dotted line uses the "medium node" of the last empty cell). We remind that, when used once, the command `\@@_create_extra_nodes:` becomes no-op in the current TeX group.

```
    \bool_if:nT { \l_@@_initial_open_bool || \l_@@_final_open_bool }
      \@@_create_extra_nodes:
  }
```

If the dotted line to draw is in the list of the previously drawn lines (`\g_@@_yet_drawn_seq`), we don't draw (so, we won't have overlapping lines in the PDF). The token list `\l_tmpa_tl` is the 4-uplet characteristic of the line.

```
\prg_set_conditional:Npnn \@@_if_yet_drawn: { F }
  {
    \tl_set:Nx \l_tmpa_tl
      {
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int -
        \int_use:N \l_@@_final_i_int -
        \int_use:N \l_@@_final_j_int
      }
    \seq_if_in:NVTF \g_@@_yet_drawn_seq \l_tmpa_tl
```

If the dotted line to draw is not in the list, we add it to the list `\g_@@_yet_drawn_seq`.

```
      \prg_return_true:
      {
        \seq_gput_left:NV \g_@@_yet_drawn_seq \l_tmpa_tl
        \prg_return_false:
      }
  }
```

The command `\@@_retrieve_coords:nn` retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw [20]. 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`.

[20] 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 prefix 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 \@@_retrieve_coords:nn
  {
    \dim_gzero_new:N \g_@@_x_initial_dim
    \dim_gzero_new:N \g_@@_y_initial_dim
    \dim_gzero_new:N \g_@@_x_final_dim
    \dim_gzero_new:N \g_@@_y_final_dim
    \begin { tikzpicture } [ remember~picture ]
      \tikz@parse@node \pgfutil@firstofone
        ( nm - \int_use:N \g_@@_env_int -
               \int_use:N \l_@@_initial_i_int -
               \int_use:N \l_@@_initial_j_int #1 )
      \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
      \dim_gset:Nn \g_@@_y_initial_dim \pgf@y
      \tikz@parse@node \pgfutil@firstofone
        ( nm - \int_use:N \g_@@_env_int -
               \int_use:N \l_@@_final_i_int -
               \int_use:N \l_@@_final_j_int #2 )
      \dim_gset:Nn \g_@@_x_final_dim \pgf@x
      \dim_gset:Nn \g_@@_y_final_dim \pgf@y
    \end { tikzpicture }
  }
\cs_generate_variant:Nn \@@_retrieve_coords:nn { x x }
```

```
\cs_new_protected:Nn \@@_draw_Ldots:nn
  {
    \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
    \@@_if_yet_drawn:F \@@_actually_draw_Ldots:
  }
```

The command `\@@_actually_draw_Ldots:` actually 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 \@@_actually_draw_Ldots:
  {
    \@@_retrieve_coords:xx
      {
        \bool_if:NTF \l_@@_initial_open_bool
          { - medium.base~west }
          { .base~east }
      }
      {
        \bool_if:NTF \l_@@_final_open_bool
          { - medium.base~east }
          { . base~west }
      }
    \bool_if:NT \l_@@_initial_open_bool
      { \dim_gset_eq:NN \g_@@_y_initial_dim \g_@@_y_final_dim }
    \bool_if:NT \l_@@_final_open_bool
      { \dim_gset_eq:NN \g_@@_y_final_dim \g_@@_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_@@_y_initial_dim { 0.53 pt }
    \dim_gadd:Nn \g_@@_y_final_dim { 0.53 pt }
    \@@_draw_tikz_line:
  }
```

```
\cs_new_protected:Nn \@@_draw_Cdots:nn
  {
    \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
    \@@_if_yet_drawn:F
      {
        \@@_retrieve_coords:xx
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { - medium.mid~west }
              { .mid~east }
          }
          {
            \bool_if:NTF \l_@@_final_open_bool
            { - medium.mid~east }
            { .mid~west }
          }
        \bool_if:NT \l_@@_initial_open_bool
          { \dim_gset_eq:NN \g_@@_y_initial_dim \g_@@_y_final_dim }
        \bool_if:NT \l_@@_final_open_bool
          { \dim_gset_eq:NN \g_@@_y_final_dim \g_@@_y_initial_dim }
        \@@_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 \@@_draw_Vdots:nn
  {
    \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_zero_int
    \@@_if_yet_drawn:F
      { \@@_retrieve_coords:xx
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { - medium.north~west }
              { .south~west }
          }
          {
            \bool_if:NTF \l_@@_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_@@_x_initial_dim = \g_@@_x_final_dim }
        \@@_retrieve_coords:xx
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { - medium.north }
              { .south }
          }
          {
            \bool_if:NTF \l_@@_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_@@_x_initial_dim = \g_@@_x_final_dim }
        \bool_if:NF \l_tmpb_bool
          {
            \dim_gset:Nn \g_@@_x_initial_dim
              {
                \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
                  \g_@@_x_initial_dim \g_@@_x_final_dim
              }
            \dim_gset_eq:NN \g_@@_x_final_dim \g_@@_x_initial_dim
          }
        \@@_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 \@@_draw_Ddots:nn
  {
    \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_one_int
    \@@_if_yet_drawn:F
      {
        \@@_retrieve_coords:xx
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { - medium.north~west }
              { .south~east }
          }
          {
            \bool_if:NTF \l_@@_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_@@_parallelize_diags_bool
          {
            \int_incr:N \l_@@_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_@@_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_@@_delta_x_one_dim
                  { \g_@@_x_final_dim - \g_@@_x_initial_dim }
                \dim_set:Nn \l_@@_delta_y_one_dim
                  { \g_@@_y_final_dim - \g_@@_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_@@_y_final_dim
                  {
                    \g_@@_y_initial_dim +
                    ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
                    \dim_ratio:nn \l_@@_delta_y_one_dim \l_@@_delta_x_one_dim
                  }
              }
          }
```

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

```
        \@@_draw_tikz_line:
      }
  }
```

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

```
\cs_new_protected:Nn \@@_draw_Iddots:nn
  {
    \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
    \@@_if_yet_drawn:F
      { \@@_retrieve_coords:xx
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { - medium.north~east }
              { .south~west }
          }
          {
            \bool_if:NTF \l_@@_final_open_bool
              { - medium.south~west }
              { .north~east }
          }
        \bool_if:NT \l_@@_parallelize_diags_bool
          {
            \int_incr:N \l_@@_iddots_int
            \int_compare:nNnTF \l_@@_iddots_int = \c_one_int
              {
                \dim_set:Nn \l_@@_delta_x_two_dim
                  { \g_@@_x_final_dim - \g_@@_x_initial_dim }
                \dim_set:Nn \l_@@_delta_y_two_dim
                  { \g_@@_y_final_dim - \g_@@_y_initial_dim }
              }
              {
                \dim_gset:Nn \g_@@_y_final_dim
                   {
                     \g_@@_y_initial_dim +
                     ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
                     \dim_ratio:nn \l_@@_delta_y_two_dim \l_@@_delta_x_two_dim
                   }
              }
          }
        \@@_draw_tikz_line:
      }
  }
```

13.6 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 \@@_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_@@_l_dim
    \dim_set:Nn \l_@@_l_dim
      {
        \fp_to_dim:n
          {
            sqrt
            ( (   \dim_use:N \g_@@_x_final_dim
                - \dim_use:N \g_@@_x_initial_dim
              ) ^ 2
                 +
              (   \dim_use:N \g_@@_y_final_dim
                - \dim_use:N \g_@@_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_@@_l_dim = \c_zero_dim
```

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

```
      {
        \bool_if:NTF \l_@@_initial_open_bool
          {
            \bool_if:NTF \l_@@_final_open_bool
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn \l_@@_l_dim { 0.45 em } }
              }
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l_@@_l_dim - 0.3 em } { 0.45 em } }
              }
          }
          {
            \bool_if:NTF \l_@@_final_open_bool
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l_@@_l_dim - 0.3 em } { 0.45 em } }
              }
              {
                \int_set:Nn \l_tmpa_int
                  { \dim_ratio:nn { \l_@@_l_dim - 0.6 em } { 0.45 em }}
              }
          }
```

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_@@_x_final_dim - \g_@@_x_initial_dim ) *
            \dim_ratio:nn { 0.45 em } \l_@@_l_dim
          }
        \dim_set:Nn \l_tmpb_dim
          {
            ( \g_@@_y_final_dim - \g_@@_y_initial_dim ) *
            \dim_ratio:nn { 0.45 em } \l_@@_l_dim
          }
```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0$, 1 or 2. We first compute this number k in `\l_tmpb_int`.

```
        \int_set:Nn \l_tmpb_int
          {
            \bool_if:NTF \l_@@_initial_open_bool
              { \bool_if:NTF \l_@@_final_open_bool 1 0 }
              { \bool_if:NTF \l_@@_final_open_bool 2 1 }
          }
```

In the loop over the dots (`\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_@@_x_initial_dim
          {
            ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
            \dim_ratio:nn
              { \l_@@_l_dim - 0.45 em * \l_tmpa_int } { \l_@@_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_@@_y_initial_dim
          {
            ( \g_@@_y_final_dim - \g_@@_y_initial_dim ) *
            \dim_ratio:nn
              { \l_@@_l_dim - 0.45 em * \l_tmpa_int }
              { \l_@@_l_dim * 2 } *
            \l_tmpb_int
          }
        \begin { tikzpicture } [ overlay ]
          \int_step_inline:nnnn 0 1 \l_tmpa_int
            {
              \pgfpathcircle
                { \pgfpoint { \g_@@_x_initial_dim } { \g_@@_y_initial_dim } }
                { 0.53 pt }
              \pgfusepath { fill }
              \dim_gadd:Nn \g_@@_x_initial_dim \l_tmpa_dim
              \dim_gadd:Nn \g_@@_y_initial_dim \l_tmpb_dim
            }
        \end { tikzpicture }
    }
}
```

13.7 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 \@@_ldots \ldots
\cs_set_eq:NN \@@_cdots \cdots
\cs_set_eq:NN \@@_vdots \vdots
\cs_set_eq:NN \@@_ddots \ddots
\cs_set_eq:NN \@@_iddots \iddots
```

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 \@@_add_to_empty_cells:
  {
    \seq_gput_right:Nx \g_@@_empty_cells_seq
      { \int_use:N \g_@@_row_int - \int_use:N \g_@@_column_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}`).

```
\NewDocumentCommand \@@_Ldots { s }
  {
    \bool_if:nF { #1 } { \@@_instruction_of_type:n { Ldots } }
    \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_ldots }
    \@@_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \@@_Cdots { s }
  {
    \bool_if:nF { #1 } { \@@_instruction_of_type:n { Cdots } }
    \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_cdots }
    \@@_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \@@_Vdots { s }
  {
    \bool_if:nF { #1 } { \@@_instruction_of_type:n { Vdots } }
    \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_vdots }
    \@@_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \@@_Ddots { s }
  {
    \bool_if:nF { #1 } { \@@_instruction_of_type:n { Ddots } }
    \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_ddots }
    \@@_add_to_empty_cells:
  }
```

```
\NewDocumentCommand \@@_Iddots { s }
  {
    \bool_if:nF { #1 } { \@@_instruction_of_type:n { Iddots } }
    \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_iddots }
    \@@_add_to_empty_cells:
  }
```

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

```
\cs_new_protected:Nn \@@_Hspace:
  {
   \@@_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 \@@_old_multicolumn \multicolumn
\cs_new:Nn \@@_multicolumn:nnn
  {
    \@@_old_multicolumn { #1 } { #2 } { #3 }
    \int_compare:nNnT #1 > 1
      {
        \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
          { \int_eval:n \g_@@_row_int - \int_use:N \g_@@_column_int }
        \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
      }
    \int_gadd:Nn \g_@@_column_int { #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 beetween two consecutive dots). Tikz nodes are created for all the cells of the array, even the implicit cells of the `\Hdotsfor`.

```
\NewDocumentCommand \@@_Hdotsfor { O { } m }
  {
    \tl_gput_right:Nx \g_@@_lines_to_draw_tl
      {
        \exp_not:N \@@_draw_Hdotsfor:nnn
          { \int_use:N \g_@@_row_int }
          { \int_use:N \g_@@_column_int }
          { #2 }
      }
    \prg_replicate:nn { #2 - 1 } { & }
  }
```

```
\cs_new_protected:Nn \@@_draw_Hdotsfor:nnn
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
    \int_set:Nn \l_@@_initial_i_int { #1 }
    \int_set:Nn \l_@@_final_i_int { #1 }
```

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

```
    \int_compare:nNnTF #2 = 1
      {
        \int_set:Nn \l_@@_initial_j_int 1
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \int_set:Nn \l_tmpa_int { #2 - 1 }
        \@@_if_not_empty_cell:nnTF \l_@@_initial_i_int \l_tmpa_int
          { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
          {
            \int_set:Nn \l_@@_initial_j_int {#2}
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #2 + #3 -1 } = \g_@@_column_int
      {
        \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      }
      {
        \int_set:Nn \l_tmpa_int { #2 + #3 }
        \@@_if_not_empty_cell:nnTF \l_@@_final_i_int \l_tmpa_int
          { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
          {
            \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
            \bool_set_true:N \l_@@_final_open_bool
          }
      }
    \bool_if:nT { \l_@@_initial_open_bool || \l_@@_final_open_bool }
      \@@_create_extra_nodes:
    \@@_actually_draw_Ldots:
  }
```

13.8 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.

```
\cs_new_protected:Nn \@@_line:nn
  {
    \dim_zero_new:N \g_@@_x_initial_dim
    \dim_zero_new:N \g_@@_y_initial_dim
    \dim_zero_new:N \g_@@_x_final_dim
    \dim_zero_new:N \g_@@_y_final_dim
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
    \begin { tikzpicture }
       \path~(#1)~--~(#2)~node[at~start]~(i)~{}~node[at~end]~(f)~{} ;
       \tikz@parse@node \pgfutil@firstofone ( i )
       \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
       \dim_gset:Nn \g_@@_y_initial_dim \pgf@y
       \tikz@parse@node \pgfutil@firstofone ( f )
       \dim_gset:Nn \g_@@_x_final_dim \pgf@x
       \dim_gset:Nn \g_@@_y_final_dim \pgf@y
    \end { tikzpicture }
    \@@_draw_tikz_line:
  }
```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

13.9 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 `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[21] as the line created by `\hline` (or `\hdashline` of arydshln). To this end, we construct a "false row" and, in this row, we create a Tikz node (`\coordinate`) that will be used to have the y-value of the line.

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

The command `\@@_hdottedline:` (which is linked to `\hdottedline` in the environment `{NiceArray}`) begins with a `\noalign` which contains a vertical skip which is part of the construction of the "false row".

Some extension, like the extension doc do a redefinition of the command `\dotfill` of LaTeX. That's why we define a command `\@@_dotfill:` as we wish.

```
\cs_set:Npn \@@_dotfill:
  { \cleaders \hbox_to_wd:nn {.44em} {\hss .\hss } \hfill \kern \c_zero_dim }
```

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

```
\cs_new:Nn \@@_hdottedline:
  {
    \noalign
      {
        \bool_gset_true:N \g_@@_extra_nodes_bool
        \cs_if_exist:cTF { @@_width_ \int_use:N \g_@@_env_int }
          { \dim_set:Nv \l_tmpa_dim { @@_width_ \int_use:N \g_@@_env_int } }
          { \dim_set:Nn \l_tmpa_dim { 5 mm } }
        \hbox_overlap_right:n
          { \hbox_to_wd:nn
              {
                \l_tmpa_dim + 2 \arraycolsep
                - \l_@@_left_margin_dim - \g_@@_right_margin_dim
              }
            \@@_dotfill:
          }
      }
  }
```

[21]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`.

```
\@@_msg_new:nn { Use~of~hdottedline~in~first~position }
  {
    You~can't~use~the~command~\token_to_str:N\hdottedline\ in~the~first~row~
    of~the~environment~\{\@currenvir\}.~But~maybe~you~have~used~l,~c~and~r~
    instead~of~L,~C,~R~in~the~preamble~of~the~array.\\
    If~you~go~on,~this~dotted~line~will~be~ignored.
  }
\@@_msg_new:nn { Use~of~hdottedline~in~first~position~bis }
  {
    You~can't~use~the~command~\token_to_str:N\hdottedline\ in~the~first~row~
    of~the~environment~\{\@currenvir\}. \\
    If~you~go~on,~this~dotted~line~will~be~ignored.
  }
```

```
\cs_new_protected:Nn \@@_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
      { \@@_error:n { Use~of~:~in~first~position } }
      {
        \@@_create_extra_nodes:
        \bool_if:NF \c_@@_draft_bool
          {
            \dim_zero_new:N \g_@@_x_initial_dim
            \dim_zero_new:N \g_@@_y_initial_dim
            \dim_zero_new:N \g_@@_x_final_dim
            \dim_zero_new:N \g_@@_y_final_dim
            \bool_set_true:N \l_@@_initial_open_bool
            \bool_set_true:N \l_@@_final_open_bool
```

In order to have the coordinates of the line to draw, we use the “large nodes”.

```
            \begin { tikzpicture } [ remember~picture ]
              \tikz@parse@node\pgfutil@firstofone
                ( 1 - #1 - large .north~east )
              \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
              \dim_gset:Nn \g_@@_y_initial_dim \pgf@y
              \tikz@parse@node\pgfutil@firstofone
                ( \int_use:N \g_@@_row_int - #1 - large .south~east )
              \dim_gset:Nn \g_@@_x_final_dim \pgf@x
              \dim_gset:Nn \g_@@_y_final_dim \pgf@y
            \end { tikzpicture }
```

However, if the `w`-nodes are created in the previous column (that is if the previous column was constructed explicitly or implicitly[22] with a letter `w`), we use the `w`-nodes to change the x-value of the nodes in order to have the dotted lines perfectly aligned when we use the environment `{NiceMatrixBlock}` with the option `auto-columns-width`.

```
            \cs_if_exist:cT
              { pgf@sh@ns@nm -\int_use:N \g_@@_env_int - 1 - #1 - w }
              {
                \begin { tikzpicture } [ remember~picture ]
                  \tikz@parse@node\pgfutil@firstofone
                    ( 1 - #1 - w .north~east )
                  \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
                  \tikz@parse@node\pgfutil@firstofone
                    ( \int_use:N \g_@@_row_int - #1 - w .south~east )
                  \dim_gset:Nn \g_@@_x_final_dim \pgf@x
                \end { tikzpicture }
                \dim_gadd:Nn \g_@@_x_initial_dim \arraycolsep
                \dim_gadd:Nn \g_@@_x_final_dim \arraycolsep
```

[22]A column is constructed implicitly with the letter `w` if the option `columns-width` is used or if the environment `{NiceMatrixBlock}` is used with the option `auto-columns-width`.

```
                }
              \@@_draw_tikz_line:
            }
        }
    }
\@@_msg_new:nn { Use~of~:~in~first~position }
  {
    You~can't~use~the~column~specifier~"\l_@@_letter_for_dotted_lines_str"~in~the~
    first~position~of~the~preamble~of~the~environment~\{\@currenvir\}. \\
    If~you~go~on,~this~dotted~line~will~be~ignored.
  }
```

13.10 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in "auto" mode.

```
\bool_new:N \l_@@_block_auto_columns_width_bool
```

As of now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
\keys_define:nn { NiceMatrix / NiceMatrixBlock }
  {
    auto-columns-width .code:n =
      {
        \bool_set_true:N \l_@@_block_auto_columns_width_bool
        \dim_gzero_new:N \g_@@_max_cell_width_dim
        \bool_set_true:N \l_@@_auto_columns_width_bool
      }
  }
```

```
\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
  {
    \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
    \int_zero_new:N \l_@@_first_env_block_int
    \int_set:Nn \l_@@_first_env_block_int { \g_@@_env_int + 1 }
  }
```

At the end of the environment `{NiceMatrixBlock}`, we write in the main `.aux` file instructions for the column width of all the environments of the block (that's why we have stored the number of the first environment of the block in the counter `\l_@@_first_env_block_int`).

```
  {
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \iow_now:Nn \@mainaux \ExplSyntaxOn
        \int_step_inline:nnnn \l_@@_first_env_block_int 1 \g_@@_env_int
          {
            \iow_now:Nx \@mainaux
              {
                \cs_gset:cpn { @@ _ max _ cell _ width _ ##1 }
                  { \dim_use:N \g_@@_max_cell_width_dim }
              }
          }
        \iow_now:Nn \@mainaux \ExplSyntaxOff
      }
  }
```

13.11 The environment {pNiceArrayC} and its variants

The code in this section can be removed without affecting the previous code.

First, we define a set of options for the environment {pNiceArrayC} and its variants. This set of keys is named NiceMatrix/NiceArrayC even though there is no environment called {NiceArrayC}.

```
\keys_define:nn { NiceMatrix / NiceArrayC }
  {
    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    unknown .code:n  = \@@_error:n { Unknown~option~for~NiceArrayC }
  }
\@@_msg_new:nnn { Unknown~option~for~NiceArrayC }
  {
    The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
    \{\@currenvir\}. \\
    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-last-col,~
    columns-width,~
    create-extra-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags~
    renew-dots~
    and~right-margin.
  }
```

In the environment {pNiceArrayC} (and its variants), the last column is composed with instructions \hbox_overlap_right:n (this instruction may be seen as the expl3 equivalent of the classical command \rlap). After the composition of the array, an horizontal skip is inserted to compensate for these overlapping boxes.

The command \@@_NiceArrayC:n will be used in {NiceArrayCwithDelims} but also in the environment {NiceArrayRCwithDelims}.

```
\cs_new_protected:Nn \@@_NiceArrayC:n
  {
    \bool_set_true:N \l_@@_exterior_column_bool
    \bool_gset_false:N \g_@@_exterior_column_found_bool
    \begin { NiceArray }
```

The beginning of the preamble is the argument of the environment {pNiceArrayC}.

```
      { #1
```

However, we add a last column with its own specification. For a cell in this last column, the first operation is to store the content of the cell in the box \l_tmpa_box. This is allowed in expl3 with the construction \hbox_set:Nw \l_tmpa_box ... \hbox_set_end:.

```
        >
          {
            \bool_gset_true:N \g_@@_exterior_column_found_bool
            \int_gincr:N \g_@@_column_int
            \int_gset:Nn \g_@@_column_total_int
              { \int_max:nn \g_@@_column_total_int \g_@@_column_int }
            \hbox_set:Nw \l_tmpa_box $ % $
              \l_@@_code_for_last_col_tl
          }
        l
```

We actualize the value of `\g_@@_width_last_col_dim` which, at the end of the array, will contain the maximal width of the cells of the last column (thus, it will be equal to the width of the last column).

```
          < { $ % $
              \hbox_set_end:
              \dim_gset:Nn \g_@@_width_last_col_dim
                 {
                   \dim_max:nn
                     \g_@@_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
                    {
                      2 \arraycolsep +
                      \l_@@_right_margin_dim +
                      \l_@@_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_@@_env_int -
                        \int_use:N \g_@@_row_int -
                        \int_use:N \g_@@_column_int ,
                      alias =
                        \str_if_empty:NF \l_@@_name_str
                          {
                            \l_@@_name_str -
                            \int_use:N \g_@@_row_int -
                            \int_use:N \g_@@_column_int
                          }
                    ]
                    { \box_use:N \l_tmpa_box } ;
                }
            }
        }
  }
```

The environments of the type of `{pNiceArrayC}` will be constructed over `{NiceArrayCwithDelims}`. The first two arguments of this environment are the left and the right delimiter.

```
\NewDocumentEnvironment { NiceArrayCwithDelims } { m m O { } m ! O { } }
  {
    \dim_gzero_new:N \g_@@_width_last_col_dim
    \keys_set:nn { NiceMatrix / NiceArrayC } { #3 , #5 }
    \bool_set_false:N \l_@@_exterior_arraycolsep_bool
    \str_set:Nn \l_@@_pos_env_str c
    \left #1
    \@@_NiceArrayC:n { #4 }
  }
  {
    \end { NiceArray }
    \right #2
    \skip_horizontal:n \g_@@_width_last_col_dim
  }
```

In the following environments, we don't use the form with `\begin{...}` and `\end{...}` because we use `\@currenvir` in the error message for an unknown option.

```
\NewDocumentEnvironment { pNiceArrayC } { }
  { \NiceArrayCwithDelims ( ) }
  { \endNiceArrayCwithDelims }
\NewDocumentEnvironment { vNiceArrayC } { }
  { \NiceArrayCwithDelims | | }
  { \endNiceArrayCwithDelims }
\NewDocumentEnvironment { VNiceArrayC } { }
  { \NiceArrayCwithDelims \|  \| }
  { \endNiceArrayCwithDelims }
\NewDocumentEnvironment { bNiceArrayC } { }
  { \NiceArrayCwithDelims [ ] }
  { \endNiceArrayCwithDelims }
\NewDocumentEnvironment { BNiceArrayC } { }
  { \NiceArrayCwithDelims \{  \} }
  { \endNiceArrayCwithDelims }
```

13.12 The environment {pNiceArrayRC}

The code in this section can be removed without affecting the previous code.

```
\keys_define:nn { NiceMatrix / NiceArrayRC }
  {
    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~option~for~NiceArrayRC }
  }
\@@_msg_new:nnn { Unknown~option~for~NiceArrayRC }
  {
    The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
     \{ \@currenvir \}. \\
    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-last-col,~
    code-for-first-row,~
    columns-width,~
    create-extra-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    left-margin,~
    name,~
    nullify-dots,~
    parallelize-diags,~
    renew-dots~
    and~right-margin.
  }
```

The first and the second argument of the environment {NiceArrayRCwithDelims} are the delimiters which will be used in the array. Usually, the final user will not use directly this environment {NiceArrayRCwithDelims} because he will use one of the variants {pNiceArrayRC}, {vNiceArrayRC}, etc.

```
\NewDocumentEnvironment { NiceArrayRCwithDelims } { m m O { } m ! O { } }
  {
    \int_zero:N \l_@@_nb_first_row_int
    \dim_gzero_new:N \g_@@_width_last_col_dim
    \keys_set:nn { NiceMatrix / NiceArrayRC } { #3 , #5 }
    \bool_set_false:N \l_@@_exterior_arraycolsep_bool
    \str_set:Nn \l_@@_pos_env_str c
    \box_clear_new:N \l_@@_the_array_box
    \hbox_set:Nw \l_@@_the_array_box
    $ % $
    \@@_NiceArrayC:n { #4 }
  }
  {
    \end { NiceArray }
    $ % $
    \hbox_set_end:
    \dim_set:Nn \l_tmpa_dim
      {
        (
          \dim_max:nn
             { 12 pt }
             { \g_@@_max_ht_row_one_dim + \g_@@_max_dp_row_zero_dim }
        )
        + \g_@@_max_ht_row_zero_dim - \g_@@_max_ht_row_one_dim
      }
    \hbox_set:Nn \l_tmpa_box
      {
        $ % $
        \left #1
        \vcenter
          {
            \skip_vertical:n { - \l_tmpa_dim }
            \box_use_drop:N \l_@@_the_array_box
          }
        \right #2
        $ % $
        \skip_horizontal:n \g_@@_width_last_col_dim
      }
    \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
    \box_use_drop:N \l_tmpa_box
    }
```

In the following environments, we don't use the form with \begin{...} and \end{...} because we use \@currenvir in the error message for an unknown option.

```
\NewDocumentEnvironment { pNiceArrayRC } { }
  { \NiceArrayRCwithDelims ( ) }
  { \endNiceArrayRCwithDelims }
```

```
\NewDocumentEnvironment { bNiceArrayRC } { }
  { \NiceArrayRCwithDelims [ ] }
  { \endNiceArrayRCwithDelims }
```

```
\NewDocumentEnvironment { vNiceArrayRC } { }
  { \NiceArrayRCwithDelims | | }
  { \endNiceArrayRCwithDelims }
```

```
\NewDocumentEnvironment { VNiceArrayRC } { }
  { \NiceArrayRCwithDelims \| \| }
  { \endNiceArrayRCwithDelims }
```

```
\NewDocumentEnvironment { BNiceArrayRC } { }
  { \NiceArrayRCwithDelims \{ \} }
  { \endNiceArrayRCwithDelims }
```

13.13 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 }
```

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 \@@_create_extra_nodes:
  {
    \begin { tikzpicture } [ remember~picture , overlay ]
      \int_step_variable:nnnNn \l_@@_nb_first_row_int 1 \g_@@_row_int \@@_i
        {
          \dim_zero_new:c { l_@@_row_\@@_i _min_dim }
          \dim_set_eq:cN { l_@@_row_\@@_i _min_dim } \c_max_dim
          \dim_zero_new:c { l_@@_row_\@@_i _max_dim }
          \dim_set:cn { l_@@_row_\@@_i _max_dim } { - \c_max_dim }
        }
      \int_step_variable:nNn \g_@@_column_total_int \@@_j
        {
          \dim_zero_new:c { l_@@_column_\@@_j _min_dim }
          \dim_set_eq:cN { l_@@_column_\@@_j _min_dim } \c_max_dim
          \dim_zero_new:c { l_@@_column_\@@_j _max_dim }
          \dim_set:cn { l_@@_column_\@@_j _max_dim } { - \c_max_dim }
        }
```

We begin the two nested loops over the rows and the columns of the array.

```
      \int_step_variable:nnNn \l_@@_nb_first_row_int \g_@@_row_int \@@_i
        {
          \int_step_variable:nNn \g_@@_column_total_int \@@_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_@@_env_int - \@@_i- \@@_j }
```

We retrieve the coordinates of the anchor `south west` of the (normal) node of the cell (i-j). They will be stored in `\pgf@x` and `\pgf@y`.

```
                  {
                    \tikz@parse@node \pgfutil@firstofone
                      ( nm - \int_use:N \g_@@_env_int
                           - \@@_i - \@@_j .south~west )
                    \dim_set:cn { l_@@_row_\@@_i _min_dim}
                      { \dim_min:vn { l_@@_row _ \@@_i _min_dim } \pgf@y }
                    \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i - \@@_j }
                      {
                        \dim_set:cn { l_@@_column _ \@@_j _ min_dim}
                          { \dim_min:vn { l_@@_column _ \@@_j _min_dim } \pgf@x }
                      }
```

We retrieve the coordinates of the anchor `north east` of the (normal) node of the cell (i-j). They will be stored in `\pgf@x` and `\pgf@y`.

```
                    \tikz@parse@node \pgfutil@firstofone
                      ( nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j .north~east )
                    \dim_set:cn { l_@@_row _ \@@_i _ max_dim }
                      { \dim_max:vn { l_@@_row _ \@@_i _ max_dim } \pgf@y }
                    \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i - \@@_j }
                      {
                        \dim_set:cn { l_@@_column _ \@@_j _ max_dim }
                          { \dim_max:vn { l_@@_column _ \@@_j _max_dim } \pgf@x }
                      }
                  }
              }
          }
```

Now, we 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 }
          \@@_create_nodes:
```

For "large nodes", the eventual "first row" and "last column" (in environments like `{pNiceArrayRC}`) don't interfer. That's why the loop over the rows will start at 1 and the loop over the columns will stop at `\g_@@_column_int` (and not `\g_@@_column_total_int`).[23]

```
          \int_set:Nn \l_@@_nb_first_row_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 { \g_@@_row_int - 1 } \@@_i
            {
              \dim_set:cn { l_@@_row _ \@@_i _ min _ dim }
                {
                  (
                    \dim_use:c { l_@@_row _ \@@_i _ min _ dim } +
                    \dim_use:c { l_@@_row _ \int_eval:n { \@@_i + 1 } _ max _ dim }
                  )
                  / 2
                }
              \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i + 1 } _ max _ dim }
                { l_@@_row_\@@_i _min_dim }
            }
          \int_step_variable:nNn { \g_@@_column_int - 1 } \@@_j
            {
              \dim_set:cn { l_@@_column _ \@@_j _ max _ dim }
                {
                  (
                    \dim_use:c
                      { l_@@_column _ \@@_j _ max _ dim } +
                    \dim_use:c
                      { l_@@_column _ \int_eval:n { \@@_j + 1 } _ min _ dim }
                  )
                  / 2
                }
              \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j + 1 } _ min _ dim }
                { l_@@_column _ \@@_j _ max _ dim }
            }
          \dim_sub:cn
            { l_@@_column _ 1 _ min _ dim }
            \g_@@_left_margin_dim
          \dim_add:cn
            { l_@@_column _ \int_use:N \g_@@_column_int _ max _ dim }
            \g_@@_right_margin_dim
```

[23]We recall that `\g_@@_column_total_int` is equal to `\g_@@_column_int` except if there is an exterior column. In this case, `\g_@@_column_total_int` is equal to `\g_@@_column_int` $+ 1$.

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

```
      \tikzset { name~suffix = -large }
      \@@_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:Nn \@@_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_@@_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_@@_env_int - 1 -
                \int_use:N \g_@@_column_int - 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 { @@_width_ \int_use:N \g_@@_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 \@@_create_nodes:
  {
    \int_step_variable:nnnNn \l_@@_nb_first_row_int 1 \g_@@_row_int \@@_i
      {
        \int_step_variable:nnnNn 1 1 \g_@@_column_total_int \@@_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 ( @@~south~west )
               at ( \dim_use:c { l_@@_column_ \@@_j _min_dim } ,
                    \dim_use:c { l_@@_row_ \@@_i _min_dim } ) ;
            \coordinate ( @@~north~east )
               at ( \dim_use:c { l_@@_column_ \@@_j _max_dim },
                    \dim_use:c { l_@@_row_ \@@_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 = ( @@~south~west ) ( @@~north~east )  ,
                inner~sep = \c_zero_dim ,
                name = nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j ,
                alias =
                  \str_if_empty:NF \g_@@_name_str
```

```
                    { \g_@@_name_str - \@@_i - \@@_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_@@_multicolumn_cells_seq
      \g_@@_multicolumn_sizes_seq
      \@@_node_for_multicolumn:nn
  }
```

```
\cs_new_protected:Npn \@@_extract_coords: #1 - #2 \q_stop
  {
    \cs_set:Npn \@@_i { #1 }
    \cs_set:Npn \@@_j { #2 }
  }
```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{`*n*`}{...}{...}` was issued in the format *i*-*j* and the second is the value of *n* (the length of the "multi-cell").

```
\cs_new_protected:Nn \@@_node_for_multicolumn:nn
  {
    \@@_extract_coords: #1 \q_stop
    \coordinate ( @@~south~west ) at
      (
        \dim_use:c { l_@@_column _ \@@_j _ min _ dim } ,
        \dim_use:c { l_@@_row _ \@@_i _ min _ dim }
      ) ;
    \coordinate ( @@~north~east ) at
      (
        \dim_use:c { l_@@_column _ \int_eval:n { \@@_j + #2 - 1 } _ max _ dim} ,
        \dim_use:c { l_@@_row _ \@@_i _ max _ dim }
      ) ;
    \node
      [
        node~contents = { } ,
        fit = ( @@~south~west ) ( @@~north~east ) ,
        inner~sep = \c_zero_dim ,
        name = nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j ,
        alias =
          \str_if_empty:NF \g_@@_name_str { \g_@@_name_str - \@@_i - \@@_j }
      ]
      ;
  }
```

13.14 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` execute the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.
Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

```
\ProcessKeysOptions {NiceMatrix}
```

13.15 Code for `seq_mapthread_function:NNN`

In `\@@_create_nodes:` (used twice in `\@@_create_extra_nodes:` to create the "medium nodes" and "large nodes"), we want to use `\seq_mapthread_function:NNN` which is in l3candidates). For security, we define a function `\@@_seq_mapthread_function:NNN`. We will delete the following code when `\seq_mapthread_function:NNN` will be in l3seq.

```
\cs_new:Npn \@@_seq_mapthread_function:NNN #1 #2 #3
  {
   \group_begin:
```

In the group, we can use `\seq_pop:NN` safely.

```
    \int_step_inline:nn { \seq_count:N #1 }
      {
        \seq_pop:NN #1 \l_tmpa_tl
        \seq_pop:NN #2 \l_tmpb_tl
        \exp_args:NVV #3 \l_tmpa_tl \l_tmpb_tl
      }
    \group_end:
  }
```

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

14 History

14.1 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.

14.2 Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types `L`, `C` and `R`.

14.3 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". New names are in lowercase and hyphens (but backward compatibility is kept).

14.4 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.

14.5 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.

14.6 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".

14.7 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[24], Tikz externalization is now deactivated in the environments of the extension nicematrix.[25]

14.8 Changes between version 2.1 and 2.1.2

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

14.9 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 column `C`), the cells in the column `C` 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`).

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

14.10 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 `https://www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end`

14.11 Changes between version 2.1.4 and 2.1.5

Compatibility with the classes revtex4-1 and revtex4-2.
Option `allow-duplicate-names`.

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

[25]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`.

14.12 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).

14.13 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`.