

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

May 27, 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 lines¹;
- a first row and a last column for labels;
- a control of the width of the columns.

$$\begin{array}{c} \textcolor{blue}{C_1} \quad \textcolor{blue}{C_2} \cdots \cdots \cdots \textcolor{blue}{C_n} \\ \left[\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array} \right] \begin{array}{c} \textcolor{blue}{L_1} \\ \textcolor{blue}{L_2} \\ \vdots \\ \textcolor{blue}{L_n} \end{array} \end{array}$$

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 2.1.5 of `nicematrix`, at the date of 2019/05/27.

¹If 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.

<code>{NiceMatrix}</code>	<code>{NiceArray}</code>	<code>{pNiceArrayC}</code>	<code>{pNiceArrayRC}</code>
<code>{pNiceMatrix}</code>		<code>{bNiceArrayC}</code>	<code>{bNiceArrayRC}</code>
<code>{bNiceMatrix}</code>		<code>{BNiceArrayC}</code>	<code>{BNiceArrayRC}</code>
<code>{BNiceMatrix}</code>		<code>{vNiceArrayC}</code>	<code>{vNiceArrayRC}</code>
<code>{vNiceMatrix}</code>		<code>{VNiceArrayC}</code>	<code>{VNiceArrayRC}</code>
<code>{VNiceMatrix}</code>		<code>{NiceArrayCwithDelims}</code>	<code>{NiceArrayRCwithDelims}</code>

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 cells³ 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}
```

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

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

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

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

²The command `\iddots`, defined in `nicematrix`, is a variant of `\ddots` with dots going forward: $\cdotp\cdotp\cdotp$. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of `unicode-math`.

³The precise definition of a “non-empty cell” is given below (cf. p. 11).

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`:

<pre>\begin{bNiceMatrix} 0 & \Cdots & \Cdots & 0 & \\ \Vdots & & & \Vdots & \\ \Vdots & & & \Vdots & \\ 0 & \Cdots & \Cdots & 0 & \\ \end{bNiceMatrix}</pre>	$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$
--	---

In the first column of this example, there are two instructions `\Vdots` but only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF⁴).

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.

<pre>\begin{bNiceMatrix} 0 & \Cdots & \Cdots* & 0 & \\ \Vdots & & & \Vdots & \\ \Vdots* & & & \Vdots* & \\ 0 & \Cdots & \Cdots* & 0 & \\ \end{bNiceMatrix}</pre>	$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$
--	---

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

<pre>\begin{bNiceMatrix} 0 & \Cdots & & 0 & \\ \Vdots & & & \Vdots & \\ & & & \Vdots & \\ 0 & & \Cdots & 0 & \\ \end{bNiceMatrix}</pre>	$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$
---	---

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

<pre>\begin{bNiceMatrix} 0 & \Cdots & \Hspace*{1cm} & 0 & \\ \Vdots & & & \Vdots & \\ 0 & \Cdots & & 0 & \\ \end{bNiceMatrix}</pre>	$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$
---	---

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

⁵Nevertheless, 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 classically with the environment `{pmatrix}`.

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

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

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

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.

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

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).

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

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}

```

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

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

```

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

```

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

The command `\hdotsfor` of `amsmath` takes an optional argument (between square brackets) which is used for fine tuning of the space between two consecutive dots. For homogeneity, `\Hdotsfor` has also an optional argument but this argument is discarded silently.

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` 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 conjunction 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 output of `nicematrix`.

```

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

```

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

⁶The 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) ;
```

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

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

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

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

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

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

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

The names of the “large nodes” are constructed by adding the suffix “`-large`” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.⁷

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

The “large nodes” of the first column and last column may appear too small for some usage. That's why it's possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.⁸

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

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

⁸The 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.

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

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.

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

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 & & \Ddots & \Ddots & ? & \\
0 & & \Cdots & 0 & a_n & \\
\end{NiceArray}\right]
```

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

⁹The column types `L`, `C` and `R` are defined locally inside `{NiceArray}` with `\newcolumntype` of `array`. This definition overrides an eventual previous definition.

¹⁰In 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:

```

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

```

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.

```

 $\begin{pNiceArrayC}{*6C|C}[nullify-dots,code-for-last-col={\scriptstyle}]$ 

$$\begin{array}{cccccc|c} 1 & 1 & 1 & \cdots & 1 & 0 & \\ 0 & 1 & 0 & \cdots & 0 & & L_2 \text{ gets } L_2-L_1 \\ 0 & 0 & 1 & \ddots & \vdots & & L_3 \text{ gets } L_3-L_1 \\ & & & \ddots & & \vdots & \\ \vdots & & & \ddots & 0 & & \\ 0 & & & \cdots & 0 & 1 & L_n \text{ gets } L_n-L_1 \end{array}$$


```

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

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.

```

 $\begin{NiceArrayCwithDelims}{\downarrow}{\downarrow}{CCC}$ 

$$\begin{array}{ccc|c} 1 & 2 & 3 & L_1 \\ 4 & 5 & 6 & L_2 \\ 7 & 8 & 9 & L_3 \end{array}$$


```

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}$
```

$$\begin{pmatrix} C_1 & C_2 & C_3 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

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}$
```

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

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}$
```

$$\begin{pmatrix} C_1 & \cdots & C_n \\ a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \left| \begin{matrix} b_1 \\ \vdots \\ b_n \end{matrix} \right.$$

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}$
```

$$\begin{pmatrix} C_1 & \cdots & C_n \\ a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \left| \begin{matrix} b_1 \\ \vdots \\ b_n \end{matrix} \right.$$

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}$
```

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \\ \hline S_1 & \cdots & S_n \end{pmatrix} \begin{matrix} L_1 \\ \vdots \\ L_n \end{matrix}$$

9 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`.

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

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

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}
```

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

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

10 Technical remarks

10.1 Diagonal lines

By default, all the diagonal lines¹² of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

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

Example with parallelization (default):

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

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

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

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

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

The same example without parallelization:

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

10.2 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 output has a width less than 0.5 pt is empty.

¹²We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in `code-after`.

- 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`.

10.3 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`.

10.4 The class option `draft`

The package `nicematrix` is rather slow when drawing the dotted lines (generated by `\Cdots`, `\Ldots`, `\Ddots`, etc.).¹⁴

That’s why, when the class option `draft` is used, the dotted lines are not drawn, for a faster compilation.

10.5 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.

<pre>\begin{pNiceMatrix} a & \frac{AB}{B} \\ b & c \end{pNiceMatrix}</pre>	$\begin{pmatrix} a & \frac{A}{B} \\ b & c \end{pmatrix}$
--	--

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

<pre>\begin{pNiceMatrix} a & \frac{AB}{B} \\ \noalign{\kern2mm} b & c \end{pNiceMatrix}</pre>	$\begin{pmatrix} a & \frac{A}{B} \\ b & c \end{pmatrix}$
---	--

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

<pre>\begin{pNiceMatrix} a & \multicolumn{1}{\frac{AB}{B}} \\ b & c \end{pNiceMatrix}</pre>	$\begin{pmatrix} a & \frac{A}{B} \\ b & c \end{pmatrix}$
---	--

¹³In 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}{@{}cccc@{}}`.

¹⁴The 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.

10.6 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.

```
\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:}
```

11 Examples

11.1 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 & & \\
\Vdots & & & & & & b      \\
0      & \Cdots & & & 0      & b      & a
\end{pNiceMatrix}
```

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

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}
```

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

An example with `\Iddots`:

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

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

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}
```

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

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}
```

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

An example for the resultant of two polynomials (the dashed line has been drawn with `arydshln`):

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

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

$$\begin{array}{c}
 \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 9 \\ 3 & 9 & 27 & 81 & 36 \\ 4 & 16 & 64 & 256 & 100 \end{array} \right) \quad \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 3 & 18 & 6 \\ 0 & 0 & -2 & -14 & -\frac{9}{2} \end{array} \right) \begin{array}{l} L_3 \leftarrow -3L_2 + L_3 \\ L_4 \leftarrow L_2 - L_4 \end{array} \\
 \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 2 & 6 & 14 & 7 \\ 0 & 6 & 24 & 78 & 33 \\ 0 & 12 & 60 & 252 & 96 \end{array} \right) \begin{array}{l} L_2 \leftarrow -2L_1 + L_2 \\ L_3 \leftarrow -3L_1 + L_3 \\ L_4 \leftarrow -4L_1 + L_4 \end{array} \quad \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & -2 & -14 & -\frac{9}{2} \end{array} \right) \begin{array}{l} L_3 \leftarrow \frac{1}{3}L_3 \end{array} \\
 \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 3 & 12 & 39 & \frac{33}{2} \\ 0 & 1 & 5 & 21 & 8 \end{array} \right) \begin{array}{l} L_2 \leftarrow \frac{1}{2}L_2 \\ L_3 \leftarrow \frac{1}{2}L_3 \\ L_4 \leftarrow \frac{1}{12}L_4 \end{array} \quad \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 7 & \frac{7}{2} \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & 0 & -2 & -\frac{1}{2} \end{array} \right) \begin{array}{l} L_4 \leftarrow 2L_3 + L_4 \end{array}
 \end{array}$$

11.3 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`.

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

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

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

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

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

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}
```

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

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

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

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

```
\begin{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}
```


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

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

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

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

We obtain the following matrix.

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

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

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

We obtain the following matrix.

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

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

```
\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)
```

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

11.4 Block matrices

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

```
\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_{43} & a_{44}
\end{NiceArray}\right)
```

$$D = \begin{pmatrix} & & & & a_{13} & a_{14} \\ & 0_{22} & & & a_{23} & a_{24} \\ & & & & a_{33} & a_{34} \\ a_{31} & a_{32} & & & a_{33} & a_{34} \\ a_{41} & a_{42} & & & a_{34} & a_{44} \end{pmatrix}$$

12 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.

12.1 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

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

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

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

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

```

4 \RequirePackage{l3keys2e}
5 \ProvidesExplPackage
6   {nicematrix}
7   {\myfiledate}
8   {\myfileversion}
9   {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.

```

10 \bool_new:N \c_@@_draft_bool
11 \DeclareOption { draft } { \bool_set_true:N \c_@@_draft_bool }
12 \DeclareOption* { }
13 \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`.

```

14 \RequirePackage { array }
15 \RequirePackage { amsmath }
16 \RequirePackage { xparse } [ 2018-10-17 ]

```

12.2 Technical definitions

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

```

17 \bool_new:N \c_@@_revtex_bool
18 \ifclassloaded { revtex4-1 }
19   { \bool_set_true:N \c_@@_revtex_bool }
20   { }
21 \ifclassloaded { revtex4-2 }
22   { \bool_set_true:N \c_@@_revtex_bool }
23   { }

24 \cs_new_protected:Nn \@@_error:n { \msg_error:nn { nicematrix } { #1 } }
25 \cs_new_protected:Nn \@@_error:nn { \msg_error:nn { nicematrix } { #1 } { #2 } }
26 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
27 \cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }
28 \cs_new_protected:Npn \@@_msg_redirect_name:nn
29   { \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.

```

30 \ProvideDocumentCommand \iddots { }
31 {
32   \mathinner
33     { \mkern 1 mu
34       \raise \p@ \hbox { . }
35       \mkern 2 mu
36       \raise 4 \p@ \hbox { . }
37       \mkern 2 mu
38       \raise 7 \p@ \vbox { \kern 7 pt \hbox { . } } \mkern 1 mu
39     }
40 }

```

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.

```
41 \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.

```
42 \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`.

```
43 \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.

```
44 \int_new:N \l_@@_nb_first_row_int
45 \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”.

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

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

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

12.3 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.

```
48 \str_new:N \l_@@_pos_env_str
49 \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}`).

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

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

```
51 \bool_new:N \l_@@_parallelize_diags_bool
52 \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.).

```
53 \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}`).

```
54 \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`.

```
55 \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 transferred 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.

```
56 \str_new:N \g_@@_name_str
57 \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.

```
58 \bool_new:N \l_@@_extra_nodes_bool
59 \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`.

```
60 \dim_new:N \l_@@_left_margin_dim
61 \dim_new:N \l_@@_right_margin_dim
62 \dim_new:N \g_@@_left_margin_dim
63 \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`.

```
64 \dim_new:N \l_@@_extra_left_margin_dim
65 \dim_new:N \l_@@_extra_right_margin_dim
66 \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).

```
67 \keys_define:nn { NiceMatrix / Global }
68 {
69   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
70   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`.

```
71   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
72   renew-dots .default:n = true ,
73   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
74   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 circumstances, the extra nodes are created automatically, for example when a dotted line has an “open” extremity.

```

75   create-extra-nodes .bool_set:N = \l_@@_extra_nodes_bool ,
76   create-extra-nodes .default:n = true,
77   left-margin .dim_set:N = \l_@@_left_margin_dim ,
78   left-margin .default:n = \arraycolsep ,
79   right-margin .dim_set:N = \l_@@_right_margin_dim ,
80   right-margin .default:n = \arraycolsep ,
81   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
82   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim
83 }

```

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

```

84 \keys_define:nn { NiceMatrix / Env }
85 {
86   columns-width .code:n =
87     \str_if_eq:nnTF { #1 } { auto }
88     { \bool_set_true:N \l_@@_auto_columns_width_bool }
89     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
90   columns-width .value_required:n = true ,
91   name .code:n =
92     \seq_if_in:NnTF \g_@@_names_seq { #1 }
93     { \@@_error:nn { Duplicate-name } { #1 } }
94     { \seq_gput_left:Nn \g_@@_names_seq { #1 } }
95     \str_set:Nn \l_@@_name_str { #1 } ,
96   name .value_required:n = true ,
97   code-after .tl_set:N = \l_@@_code_after_tl ,
98   code-after .initial:n = \c_empty_tl ,
99   code-after .value_required:n = true ,
100 }

```

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

```

101 \keys_define:nn { NiceMatrix }
102 {
103   NiceMatrixOptions .inherit:n = NiceMatrix / Global ,
104   NiceMatrix .inherit:n =
105     {
106       NiceMatrix / Global ,
107       NiceMatrix / Env
108     } ,
109   NiceArray .inherit:n =
110     {
111       NiceMatrix / Global ,
112       NiceMatrix / Env
113     } ,
114   NiceArrayC .inherit:n =
115     {
116       NiceMatrix / Global ,
117       NiceMatrix / Env
118     } ,
119   NiceArrayRC .inherit:n =
120     {
121       NiceMatrix / Global ,
122       NiceMatrix / Env
123     }
124 }

```

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

```

125 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
126 {

```

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

```

127   renew-matrix .code:n = \@@_renew_matrix: ,
128   renew-matrix .value_forbidden:n = true ,
129   RenewMatrix .meta:n = renew-matrix ,
130   transparent .meta:n = { renew-dots , renew-matrix } ,
131   transparent .value_forbidden:n = true,
132   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

```

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

```

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

```

135   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
136   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}`.

```

137   exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
138   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.

```

139   columns-width .code:n =
140     \str_if_eq:nnTF { #1 } { auto }
141     { \@@_error:n { Option~auto~for~columns-width } }
142     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

143   allow-duplicate-names .code:n =
144     \@@_msg_redirect_name:nn { Duplicate-name } {none} ,
145   allow-duplicate-names .value_forbidden:n = true ,

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

147 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
148 {
149   The~key~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~command~
150   \token_to_str:N \NiceMatrixOptions. \\\
151   If~you~go~on,~it~will~be~ignored. \\\
152   For~a~list~of~the~available~keys,~type~H~<return>.
153 }
154 {
155   The~available~keys~are~(in~alphabetic~order):~
156   allow-duplicate-names,~
157   code-for-last-col,~
158   exterior-arraycolsep,~
159   left-margin,~
160   nullify-dots,~
161   parallelize-diags,~
162   renew-dots,~
163   renew-matrix,~
164   right-margin,~
165   and~transparent
166 }

```

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

```

167 \@@_msg_new:nn { Option~auto~for~columns~width }
168 {
169     You~can't~give~the~value~'auto'~to~the~option~'columns~width'~here.~
170     If~you~go~on,~the~option~will~be~ignored.
171 }

```

`\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.

```

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

```

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

```

174 \keys_define:nn { NiceMatrix / NiceMatrix }
175 { unknown .code:n = \@@_error:n { Unknown~option~for~NiceMatrix } }
176 \@@_msg_new:nnn { Unknown~option~for~NiceMatrix }
177 {
178     The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
179     \{NiceMatrix\}~and~its~variants. \\
180     If~you~go~on,~it~will~be~ignored. \\
181     For~a~list~of~the~available~options,~type~H~<return>.
182 }
183 {
184     The~available~options~are~(in~alphabetic~order):~
185     code~after,~
186     columns~width,~
187     create~extra~nodes,~
188     extra~left~margin,~
189     extra~right~margin,~
190     left~margin,~
191     name,~
192     nullify~dots,~
193     parallelize~diags,~
194     renew~dots~
195     and~right~margin.
196 }

```

```

197 \@@_msg_new:nnn { Duplicate~name }
198 {
199     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
200     the~same~environment~name~twice.~You~can~go~on,~but,~
201     maybe,~you~will~have~incorrect~results~especially~
202     if~you~use~'columns~width=auto'. \\
203     For~a~list~of~the~names~already~used,~type~H~<return>. \\
204     If~you~don't~want~to~see~this~message~again,~use~the~option~
205     'allow~duplicate~names'.
206 }
207 {
208     The~names~already~defined~in~this~document~are:~
209     \seq_use:Nnnn \g_@@_names_seq { ,~ } { ,~ } { ~and~ }.
210 }

```

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

```

211 \keys_define:nn { NiceMatrix / NiceArray }
212 {

```


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

```

213   c .code:n = \str_set:Nn \l_@@_pos_env_str c ,
214   t .code:n = \str_set:Nn \l_@@_pos_env_str t ,
215   b .code:n = \str_set:Nn \l_@@_pos_env_str b ,
216   unknown .code:n = @@_error:n { Unknown-option-for~NiceArray }
217 }

218 \@@_msg_new:nnn { Unknown-option-for~NiceArray }
219 {
220   The-option-'\tl_use:N\l_keys_key_tl'~is-unknown-for-the~environment~
221   \{NiceArray\}. \\
222   If-you-go-on,~it~will~be~ignored. \\
223   For~a~list~of~the~available~options,~type~H-<return>.
224 }
225 {
226   The~available~options~are~(in~alphabetic~order):~
227   b,~
228   c,~
229   code-after,~
230   create-extra-nodes,~
231   columns-width,~
232   extra-left-margin,~
233   extra-right-margin,~
234   left-margin,~
235   name,~
236   nullify-dots,~
237   parallelize-diags,~
238   renew-dots,~
239   right-margin,~
240   and~t.
241 }

```

12.4 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}`).

```

242 \cs_new_protected:Nn \@@_Cell:
243 {

```

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

```

244   \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.

```

245   \int_compare:nNnT \g_@@_column_int = \c_one_int
246     { \int_gincr:N \g_@@_row_int }
247   \int_gset:Nn \g_@@_column_total_int
248     { \int_max:nn \g_@@_column_total_int \g_@@_column_int }
249   \hbox_set:Nw \l_tmpa_box $ % $
250   \int_compare:nNnT \g_@@_row_int = \c_zero_int
251     \l_@@_code_for_first_row_tl
252 }

253 \cs_new_protected:Nn \@@_end_Cell:
254 { $ % $
255   \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}`).

```

256   \dim_gset:Nn \g_@@_max_cell_width_dim
257     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_tmpa_box } }

```

```

258 \int_compare:nNnT \g_@@_row_int = \c_zero_int
259 {
260   \dim_gset:Nn \g_@@_max_dp_row_zero_dim
261   { \dim_max:nn \g_@@_max_dp_row_zero_dim { \box_dp:N \l_tmpa_box } }
262   \dim_gset:Nn \g_@@_max_ht_row_zero_dim
263   { \dim_max:nn \g_@@_max_ht_row_zero_dim { \box_ht:N \l_tmpa_box } }
264 }
265 \int_compare:nNnT \g_@@_row_int = \c_one_int
266 {
267   \dim_gset:Nn \g_@@_max_ht_row_one_dim
268   { \dim_max:nn \g_@@_max_ht_row_one_dim { \box_ht:N \l_tmpa_box } }
269 }

```

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

```

270 \tikz
271 [
272   remember~picture ,
273   inner~sep = \c_zero_dim ,
274   minimum~width = \c_zero_dim ,
275   baseline
276 ]
277 \node
278 [
279   anchor = base ,
280   name = nm - \int_use:N \g_@@_env_int -
281           \int_use:N \g_@@_row_int -
282           \int_use:N \g_@@_column_int ,
283   alias =
284     \str_if_empty:NF \l_@@_name_str
285     {
286       \l_@@_name_str -
287       \int_use:N \g_@@_row_int -
288       \int_use:N \g_@@_column_int
289     }
290 ]
291 \bgroup
292 \box_use:N \l_tmpa_box
293 \egroup ;
294 }

```

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 programming 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}`:

```

295 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
296 {
297   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
298   \str_set:Nn \l_@@_pos_env_str c
299   \bool_set_false:N \l_@@_exterior_arraycolsep_bool
300   \NiceArray { * \c@MaxMatrixCols C }
301 }
302 {

```

```

303 \endarray
304 \skip_horizontal:n
305 { \g_@@_right_margin_dim + \g_@@_extra_right_margin_dim - \arraycolsep }
306 }

```

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}`).

```

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

308 \@@_msg_new:nn { Yet~in~NiceArray }
309 {
310   Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~
311   nested.~We~can~go~on,~but,~maybe,~you~will~have~errors~or~an~incorrect~
312   result.
313 }

```

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

```

314 \cs_new_protected:Npn \@@_define_dots:
315 {
316   \cs_set_eq:NN \Ldots \@@_Ldots
317   \cs_set_eq:NN \Cdots \@@_Cdots
318   \cs_set_eq:NN \Vdots \@@_Vdots
319   \cs_set_eq:NN \Ddots \@@_Ddots
320   \cs_set_eq:NN \Iddots \@@_Iddots
321   \bool_if:NT \l_@@_renew_dots_bool
322   {
323     \cs_set_eq:NN \ldots \@@_Ldots
324     \cs_set_eq:NN \cdots \@@_Cdots
325     \cs_set_eq:NN \vdots \@@_Vdots
326     \cs_set_eq:NN \ddots \@@_Ddots
327     \cs_set_eq:NN \iddots \@@_Iddots
328     \cs_set_eq:NN \dots \@@_Ldots
329     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor
330   }
331 }

```

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.

```

332 \cs_new_protected:Npn \@@_define_dots_to_nil:
333 {
334   \cs_set_eq:NN \Ldots \prg_do_nothing:
335   \cs_set_eq:NN \Cdots \prg_do_nothing:
336   \cs_set_eq:NN \Vdots \prg_do_nothing:
337   \cs_set_eq:NN \Ddots \prg_do_nothing:
338   \cs_set_eq:NN \Iddots \prg_do_nothing:
339   \bool_if:NT \l_@@_renew_dots_bool
340   {
341     \cs_set_eq:NN \ldots \prg_do_nothing:
342     \cs_set_eq:NN \cdots \prg_do_nothing:
343     \cs_set_eq:NN \vdots \prg_do_nothing:
344     \cs_set_eq:NN \ddots \prg_do_nothing:
345     \cs_set_eq:NN \iddots \prg_do_nothing:

```

```

346     \cs_set_eq:NN \dots \prg_do_nothing:
347     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor
348 }
349 }

```

First, we test if we are yet in an environment `{NiceArray}` (nested environments are forbidden).

```

350 \NewDocumentEnvironment { NiceArray } { 0 { } m ! 0 { } }
351 {
352     \bool_if:NT \l_@@_in_NiceArray_bool
353     { \@@_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).

```

354     \cs_if_exist:NT \tikz@library@external@loaded
355     { \tikzset { external / export = false } }
356     \bool_set_true:N \l_@@_in_NiceArray_bool
357     \group_insert_after:N \@@_after_array:
358     \tl_gclear_new:N \g_@@_lines_to_draw_tl

```

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

```

359     \int_gincr:N \g_@@_env_int
360     \bool_if:NF \l_@@_block_auto_columns_width_bool
361     { \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.

```

362     \dim_gzero_new:N \g_@@_max_dp_row_zero_dim
363     \dim_gzero_new:N \g_@@_max_ht_row_zero_dim
364     \dim_gzero_new:N \g_@@_max_ht_row_one_dim
365     \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).

```

366     \bool_if:NT \l_@@_auto_columns_width_bool
367     {
368         \group_insert_after:N \@@_write_max_cell_width:
369         \cs_if_free:cTF { _@@_max_cell_width_ \int_use:N \g_@@_env_int }
370         { \dim_set:Nn \l_@@_columns_width_dim \c_zero_dim }
371         {
372             \dim_set:Nx \l_@@_columns_width_dim
373             { \use:c { _@@_max_cell_width_ \int_use:N \g_@@_env_int } }
374         }

```

If the environment has a name, we read the value of the maximal value of the columns from `_@@_name_cell_widthname` (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)).

```

375     \str_if_empty:NF \l_@@_name_str
376     {
377         \cs_if_free:cF { _@@_max_cell_width_ \l_@@_name_str }
378         {
379             \dim_set:Nx \l_@@_columns_width_dim
380             { \use:c { _@@_max_cell_width_ \l_@@_name_str } }
381         }
382     }
383 }

```

We don't want to patch any code and that's why some code is excuted 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.

```

384     \bool_gset_eq:NN \g_@@_extra_nodes_bool \l_@@_extra_nodes_bool
385     \dim_gset_eq:NN \g_@@_left_margin_dim \l_@@_left_margin_dim

```

```

386 \dim_gset_eq:NN \g_@@_right_margin_dim \l_@@_right_margin_dim
387 \dim_gset_eq:NN \g_@@_extra_right_margin_dim \l_@@_extra_right_margin_dim
388 \tl_gset_eq:NN \g_@@_code_after_tl \l_@@_code_after_tl
389 \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

```

390 \cs_set:Npn \ialign
391 {
392   \everycr { \noalign { \int_gzero:N \g_@@_column_int } }
393   \tabskip = \c_zero_skip
394   \cs_set:Npn \ialign
395   {
396     \everycr { }
397     \tabskip = \c_zero_skip
398     \halign
399   }
400   \halign
401 }

```

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}`.

```

402 \dim_compare:nNnTF \l_@@_columns_width_dim = \c_zero_dim
403 {
404   \newcolumnntype L { > \@@_Cell: l < \@@_end_Cell: }
405   \newcolumnntype C { > \@@_Cell: c < \@@_end_Cell: }
406   \newcolumnntype R { > \@@_Cell: r < \@@_end_Cell: }
407 }

```

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.

```

408 {
409   \newcolumnntype L { w l { \dim_use:N \l_@@_columns_width_dim } }
410   \newcolumnntype C { w c { \dim_use:N \l_@@_columns_width_dim } }
411   \newcolumnntype R { w r { \dim_use:N \l_@@_columns_width_dim } }
412 }

```

We nullify the definitions of the column types `w` and `W` 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:`.

```

413 \cs_set_eq:NN \NC@find@w \relax
414 \cs_set_eq:NN \NC@find@W \relax

```

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

```

415 \newcolumnntype w [ 2 ]
416 {
417   > {
418     \hbox_set:Nw \l_tmpa_box
419     \@@_Cell:
420   }
421   < {
422     \@@_end_Cell:
423     \hbox_set_end:
424     \makebox [ ##2 ] [ ##1 ] { \box_use:N \l_tmpa_box }
425   }
426 }
427 \newcolumnntype W [ 2 ]
428 {
429   > {
430     \hbox_set:Nw \l_tmpa_box
431     \@@_Cell:

```

¹⁸With this programming, 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}`)

```

432     }
433     c
434     < {
435         \@@_end_Cell:
436         \hbox_set_end:
437         \cs_set_eq:NN \hss \hfil
438         \makebox [ ##2 ] [ ##1 ] { \box_use:N \l_tmpa_box }
439     }
440 }

```

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).

```

441 \bool_if:NTF \c_@@_draft_bool
442   \@@_define_dots_to_nil:
443   \@@_define_dots:
444   \cs_set_eq:NN \Hspace \@@_Hspace:
445   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor
446   \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.

```

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

```

448 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
449 \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.

```

450 \int_gzero_new:N \g_@@_row_int
451 \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.

```

452 \int_gzero_new:N \g_@@_column_int
453 \int_gzero_new:N \g_@@_column_total_int
454 \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.

```

455 \bool_if:NF \l_@@_exterior_arraycolsep_bool
456   { \skip_horizontal:n { - \arraycolsep } }
457   \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`.

```

458 \bool_if:NTF \c_@@_revtex_bool
459   {
460     \cs_set_eq:NN \@acol1 \@arrayacol
461     \cs_set_eq:NN \@acolr \@arrayacol
462     \cs_set_eq:NN \@acol \@arrayacol
463     \cs_set:Npn \@halignto { }

```

```

464     \array@array
465   }
466   \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`.

```

467   [ \l_@@_pos_env_str ] { #2 }
468 }

469 { \endarray
470   \bool_if:NF \l_@@_exterior_arraycolsep_bool
471     { \skip_horizontal:n { - \arraycolsep } }
472   \skip_horizontal:n
473     { \g_@@_right_margin_dim + \g_@@_extra_right_margin_dim }
474 }

```

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

```

475 \NewDocumentEnvironment { pNiceMatrix } { }
476 { \left( \begin{NiceMatrix} }
477 { \end{NiceMatrix} \right) }

478 \NewDocumentEnvironment { bNiceMatrix } { }
479 { \left[ \begin{NiceMatrix} }
480 { \end{NiceMatrix} \right] }

481 \NewDocumentEnvironment { BNiceMatrix } { }
482 { \left\{ \begin{NiceMatrix} }
483 { \end{NiceMatrix} \right\} }

484 \NewDocumentEnvironment { vNiceMatrix } { }
485 { \left\lvert \begin{NiceMatrix} }
486 { \end{NiceMatrix} \right\rvert }

487 \NewDocumentEnvironment { VNiceMatrix } {}
488 { \left\lVert \begin{NiceMatrix} }
489 { \end{NiceMatrix} \right\lVert }

```

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`.

```

490 \cs_new_protected:Nn \@@_write_max_cell_width:
491 {
492   \bool_if:NF \l_@@_block_auto_columns_width_bool
493   {
494     \iow_now:Nn \@mainaux \ExplSyntaxOn
495     \iow_now:Nx \@mainaux
496     {
497       \cs_gset:cpn { @@_max_cell_width_ \int_use:N \g_@@_env_int }
498       { \dim_use:N \g_@@_max_cell_width_dim }
499     }
500   }

```

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

```

500   \iow_now:Nx \@mainaux
501   {
502     \cs_gset:cpn { @@_max_cell_width_ \g_@@_name_str }
503     { \dim_use:N \g_@@_max_cell_width_dim }
504   }
505   \iow_now:Nn \@mainaux \ExplSyntaxOff
506 }
507 }

```

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.

```
508 \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 correspondent Tikz node.

```
509 {
510   \cs_if_free:cTF
511   { pgf@sh@ns@nm -\int_use:N \g_@@_env_int - \int_use:N #1 - \int_use:N #2 }
512   \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 explicitly declared empty for some reason. It's easy to check if the current cell is in this list.

```
513 {
514   \seq_if_in:NxTF \g_@@_empty_cells_seq { \int_use:N #1 - \int_use:N #2 }
515   \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.

```
516 {
517   \begin { pgfpicture }
```

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

```
518   \tl_set:Nx \l_tmpa_tl
519   { nm - \int_use:N \g_@@_env_int - \int_use:N #1 - \int_use:N #2 }
520   \pgfpointanchor \l_tmpa_tl { east }
521   \dim_gset:Nn \g_tmpa_dim \pgf@x
522   \pgfpointanchor \l_tmpa_tl { west }
523   \dim_gset:Nn \g_tmpb_dim \pgf@x
524   \end { pgfpicture }
525   \dim_compare:nTF
526   { \dim_abs:n { \g_tmpb_dim - \g_tmpa_dim } < 0.5 pt }
527   \prg_return_false:
528   \prg_return_true:
529 }
530 }
531 }
```

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}
```

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

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

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

```
532 \cs_new_protected:Nn \@@_instruction_of_type:n
533 {
534   \tl_gput_right:Nx \g_@@_lines_to_draw_tl
535   {
536     \exp_not:c { @@ _ draw _ #1 : nn }
537     { \int_use:N \g_@@_row_int }
538     { \int_use:N \g_@@_column_int }
539   }
540 }
```


12.5 After the construction of the array

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

```

541 \cs_new_protected:Nn \@@_after_array:
542 {
543   \group_begin:
544   \cs_if_exist:NT \tikz@library@external@loaded
545   { \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.

```

546   \int_gset_eq:NN \g_@@_column_int \g_@@_column_total_int
547   \bool_if:NT \l_@@_exterior_column_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.

```

548   \seq_gclear_new:N \g_@@_yet_drawn_seq

```

By default, the diagonal lines will be parallelized¹⁹. 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.

```

549   \bool_if:NT \l_@@_parallelize_diags_bool
550   {
551     \int_zero_new:N \l_@@_ddots_int
552     \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.

```

553     \dim_zero_new:N \l_@@_delta_x_one_dim
554     \dim_zero_new:N \l_@@_delta_y_one_dim
555     \dim_zero_new:N \l_@@_delta_x_two_dim
556     \dim_zero_new:N \l_@@_delta_y_two_dim
557   }

```

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).

```

558   \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`.

```

559   \tl_if_empty:NF \g_@@_lines_to_draw_tl
560   {
561     \int_zero_new:N \l_@@_initial_i_int
562     \int_zero_new:N \l_@@_initial_j_int
563     \int_zero_new:N \l_@@_final_i_int
564     \int_zero_new:N \l_@@_final_j_int
565     \bool_set_false:N \l_@@_initial_open_bool
566     \bool_set_false:N \l_@@_final_open_bool
567     \g_@@_lines_to_draw_tl
568   }
569   \tl_gclear:N \g_@@_lines_to_draw_tl

```

Now, the code-after.

```

570   \tikzset
571   {

```

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

```

572     every~picture / .style =
573     {
574         overlay ,
575         remember~picture ,
576         name~prefix = nm - \int_use:N \g_@@_env_int -
577     }
578 }
579 \cs_set_eq:NN \line \@@_line:nn
580 \g_@@_code_after_tl
581 \group_end:
582 }

```

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

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).

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

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.

```

583 \cs_new_protected:Nn \@@_find_extremities_of_line:nnnn
584 {
585     \int_set:Nn \l_@@_initial_i_int { #1 }
586     \int_set:Nn \l_@@_initial_j_int { #2 }
587     \int_set:Nn \l_@@_final_i_int { #1 }
588     \int_set:Nn \l_@@_final_j_int { #2 }
589     \bool_set_false:N \l_@@_initial_open_bool
590     \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 `\1 @@ stop loop bool` will be used to control these loops.

```

591     \bool_set_false:N \l_@@_stop_loop_bool
592     \bool_do_until:Nn \l_@@_stop_loop_bool
593     {
594         \int_add:Nn \l_@@_final_i_int { #3 }
595         \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

596     \bool_if:nTF
597     {
598         \int_compare_p:nNn
599         \l_@@_final_i_int < { \l_@@_nb_first_row_int - 1 }
600         || \int_compare_p:nNn \l_@@_final_i_int > \g_@@_row_int
601         || \int_compare_p:nNn \l_@@_final_j_int < \c_one_int
602         || \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).

```

603         || \int_compare_p:nNn \l_@@_final_j_int > \g_@@_column_int
604         && \int_compare_p:nNn { #4 } > \c_zero_int
605     }

```

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

```

606     {
607         \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).

```

608         \int_sub:Nn \l_@@_final_i_int { #3 }
609         \int_sub:Nn \l_@@_final_j_int { #4 }
610         \bool_set_true:N \l_@@_stop_loop_bool
611     }

```

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.

```

612     {
613         @@_if_not_empty_cell:nnT \l_@@_final_i_int \l_@@_final_j_int
614         { \bool_set_true:N \l_@@_stop_loop_bool }
615     }
616 }

```

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

```

617     \bool_set_false:N \l_@@_stop_loop_bool
618     \bool_do_until:Nn \l_@@_stop_loop_bool
619     {
620         \int_sub:Nn \l_@@_initial_i_int { #3 }
621         \int_sub:Nn \l_@@_initial_j_int { #4 }
622         \bool_if:nTF
623         {
624             \int_compare_p:nNn \l_@@_initial_i_int < \l_@@_nb_first_row_int
625             ||
626             \int_compare_p:nNn \l_@@_initial_i_int > \g_@@_row_int
627             ||
628             \int_compare_p:nNn \l_@@_initial_j_int < 1
629             ||
630             \int_compare_p:nNn \l_@@_initial_j_int > \g_@@_column_total_int
631         }
632         {
633             \bool_set_true:N \l_@@_initial_open_bool
634             \int_add:Nn \l_@@_initial_i_int { #3 }
635             \int_add:Nn \l_@@_initial_j_int { #4 }
636             \bool_set_true:N \l_@@_stop_loop_bool
637         }
638         {
639             @@_if_not_empty_cell:nnT
640             \l_@@_initial_i_int \l_@@_initial_j_int
641             { \bool_set_true:N \l_@@_stop_loop_bool }
642         }
643     }

```

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.

```

644   \bool_if:nT { \l_@@_initial_open_bool || \l_@@_final_open_bool }
645   \@@_create_extra_nodes:
646 }

```

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.

```

647 \prg_set_conditional:Npnn \@@_if_yet_drawn: { F }
648 {
649   \tl_set:Nx \l_tmpa_tl
650   {
651     \int_use:N \l_@@_initial_i_int -
652     \int_use:N \l_@@_initial_j_int -
653     \int_use:N \l_@@_final_i_int -
654     \int_use:N \l_@@_final_j_int
655   }
656   \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`.

```

657   \prg_return_true:
658   {
659     \seq_gput_left:NV \g_@@_yet_drawn_seq \l_tmpa_tl
660   \prg_return_false:
661   }
662 }

```

The command `\@@_retrieve_coords:nn` retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw²⁰. 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`. 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}`.

```

663 \cs_new_protected:Nn \@@_retrieve_coords:nn
664 {
665   \dim_gzero_new:N \g_@@_x_initial_dim
666   \dim_gzero_new:N \g_@@_y_initial_dim
667   \dim_gzero_new:N \g_@@_x_final_dim
668   \dim_gzero_new:N \g_@@_y_final_dim
669   \begin { tikzpicture } [ remember-picture ]
670     \tikz@parse@node\pgfutil@firstofone
671     ( nm - \int_use:N \g_@@_env_int -
672       \int_use:N \l_@@_initial_i_int -
673       \int_use:N \l_@@_initial_j_int #1 )
674     \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
675     \dim_gset:Nn \g_@@_y_initial_dim \pgf@y
676     \tikz@parse@node\pgfutil@firstofone
677     ( nm - \int_use:N \g_@@_env_int -
678       \int_use:N \l_@@_final_i_int -
679       \int_use:N \l_@@_final_j_int #2 )
680     \dim_gset:Nn \g_@@_x_final_dim \pgf@x
681     \dim_gset:Nn \g_@@_y_final_dim \pgf@y
682   \end { tikzpicture }
683 }
684 \cs_generate_variant:Nn \@@_retrieve_coords:nn { x x }

```

²⁰In fact, with diagonal lines, or vertical lines in columns of type L or R, an adjustment of one of the coordinates may be done.

```

685 \cs_new_protected:Nn \@@_draw_Ldots:nn
686 {
687   \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
688   \@@_if_yet_drawn:F \@@_actually_draw_Ldots:
689 }

```

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 it is used also by `\Hdotsfor`.

```

690 \cs_new_protected:Nn \@@_actually_draw_Ldots:
691 {
692   \@@_retrieve_coords:xx
693   {
694     \bool_if:NTF \l_@@_initial_open_bool
695     { - medium.base~west }
696     { .base~east }
697   }
698   {
699     \bool_if:NTF \l_@@_final_open_bool
700     { - medium.base~east }
701     { . base~west }
702   }
703   \bool_if:NT \l_@@_initial_open_bool
704   { \dim_gset_eq:NN \g_@@_y_initial_dim \g_@@_y_final_dim }
705   \bool_if:NT \l_@@_final_open_bool
706   { \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 text.

```

707   \dim_gadd:Nn \g_@@_y_initial_dim { 0.53 pt }
708   \dim_gadd:Nn \g_@@_y_final_dim { 0.53 pt }
709   \@@_draw_tikz_line:
710 }

```

```

711 \cs_new_protected:Nn \@@_draw_Cdots:nn
712 {
713   \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_zero_int \c_one_int
714   \@@_if_yet_drawn:F
715   {
716     \@@_retrieve_coords:xx
717     {
718       \bool_if:NTF \l_@@_initial_open_bool
719       { - medium.mid~west }
720       { .mid~east }
721     }
722     {
723       \bool_if:NTF \l_@@_final_open_bool
724       { - medium.mid~east }
725       { .mid~west }
726     }
727     \bool_if:NT \l_@@_initial_open_bool
728     { \dim_gset_eq:NN \g_@@_y_initial_dim \g_@@_y_final_dim }
729     \bool_if:NT \l_@@_final_open_bool
730     { \dim_gset_eq:NN \g_@@_y_final_dim \g_@@_y_initial_dim }
731     \@@_draw_tikz_line:
732   }
733 }

```

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?).

```

734 \cs_new_protected:Nn \@@_draw_Vdots:nn

```

```

735 {
736   \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_zero_int
737   \@@_if_yet_drawn:F
738   { \@@_retrieve_coords:xx
739     {
740       \bool_if:NTF \l_@@_initial_open_bool
741       { - medium.north~west }
742       { .south~west }
743     }
744     {
745       \bool_if:NTF \l_@@_final_open_bool
746       { - medium.south~west }
747       { .north~west }
748     }

```

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

```

749   \bool_set:Nn \l_tmpa_bool
750   { \dim_compare_p:nNn \g_@@_x_initial_dim = \g_@@_x_final_dim }
751   \@@_retrieve_coords:xx
752   {
753     \bool_if:NTF \l_@@_initial_open_bool
754     { - medium.north }
755     { .south }
756   }
757   {
758     \bool_if:NTF \l_@@_final_open_bool
759     { - medium.south }
760     { .north }
761   }

```

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

```

762   \bool_set:Nn \l_tmpb_bool
763   { \dim_compare_p:nNn \g_@@_x_initial_dim = \g_@@_x_final_dim }
764   \bool_if:NF \l_tmpb_bool
765   {
766     \dim_gset:Nn \g_@@_x_initial_dim
767     {
768       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
769       \g_@@_x_initial_dim \g_@@_x_final_dim
770     }
771     \dim_gset_eq:NN \g_@@_x_final_dim \g_@@_x_initial_dim
772   }
773   \@@_draw_tikz_line:
774 }
775 }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonal lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

```

776 \cs_new_protected:Nn \@@_draw_Ddots:nn
777 {
778   \@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_one_int
779   \@@_if_yet_drawn:F
780   {
781     \@@_retrieve_coords:xx
782     {
783       \bool_if:NTF \l_@@_initial_open_bool
784       { - medium.north~west }
785       { .south~east }
786     }
787     {

```

```

788         \bool_if:NTF \l_@@_final_open_bool
789         { - medium.south~east }
790         { .north~west }
791     }

```

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.

```

792     \bool_if:NT \l_@@_parallelize_diags_bool
793     {
794         \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).

```

795         \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.

```

796         {
797             \dim_set:Nn \l_@@_delta_x_one_dim
798             { \g_@@_x_final_dim - \g_@@_x_initial_dim }
799             \dim_set:Nn \l_@@_delta_y_one_dim
800             { \g_@@_y_final_dim - \g_@@_y_initial_dim }
801         }

```

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`.

```

802         {
803             \dim_gset:Nn \g_@@_y_final_dim
804             {
805                 \g_@@_y_initial_dim +
806                 ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
807                 \dim_ratio:nn \l_@@_delta_y_one_dim \l_@@_delta_x_one_dim
808             }
809         }
810     }

```

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

```

811     \@@_draw_tikz_line:
812 }
813 }

```

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

```

814 \cs_new_protected:Nn \@@_draw_Iddots:nn
815 {
816     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
817     \@@_if_yet_drawn:F
818     { \@@_retrieve_coords:xx
819         {
820             \bool_if:NTF \l_@@_initial_open_bool
821             { - medium.north~east }
822             { .south~west }
823         }
824         {
825             \bool_if:NTF \l_@@_final_open_bool
826             { - medium.south~west }
827             { .north~east }
828         }
829         \bool_if:NT \l_@@_parallelize_diags_bool
830         {
831             \int_incr:N \l_@@_iddots_int
832             \int_compare:nNnTF \l_@@_iddots_int = \c_one_int
833             {
834                 \dim_set:Nn \l_@@_delta_x_two_dim
835                 { \g_@@_x_final_dim - \g_@@_x_initial_dim }

```

```

836         \dim_set:Nn \l_@@_delta_y_two_dim
837         { \g_@@_y_final_dim - \g_@@_y_initial_dim }
838     }
839     {
840         \dim_gset:Nn \g_@@_y_final_dim
841         {
842             \g_@@_y_initial_dim +
843             ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
844             \dim_ratio:nn \l_@@_delta_y_two_dim \l_@@_delta_x_two_dim
845         }
846     }
847 }
848 \@@_draw_tikz_line:
849 }
850 }

```

12.6 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`.

```

851 \cs_new_protected:Nn \@@_draw_tikz_line:
852 {

```

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.

```

853     \dim_zero_new:N \l_@@_l_dim
854     \dim_set:Nn \l_@@_l_dim
855     {
856         \fp_to_dim:n
857         {
858             sqrt
859             ( ( \dim_use:N \g_@@_x_final_dim
860               - \dim_use:N \g_@@_x_initial_dim
861             ) ^ 2
862             +
863             ( \dim_use:N \g_@@_y_final_dim
864               - \dim_use:N \g_@@_y_initial_dim
865             ) ^ 2
866         )
867     }
868 }

```

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

```

869     \dim_compare:nNnF \l_@@_l_dim = \c_zero_dim

```

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

```

870     {
871         \bool_if:NTF \l_@@_initial_open_bool
872         {
873             \bool_if:NTF \l_@@_final_open_bool
874             {
875                 \int_set:Nn \l_tmpa_int
876                 { \dim_ratio:nn \l_@@_l_dim { 0.45 em } }
877             }
878             {
879                 \int_set:Nn \l_tmpa_int
880                 { \dim_ratio:nn { \l_@@_l_dim - 0.3 em } { 0.45 em } }
881             }
882         }
883     {

```



```

884         \bool_if:NTF \l_@@_final_open_bool
885         {
886             \int_set:Nn \l_tmpa_int
887             { \dim_ratio:nn { \l_@@_l_dim - 0.3 em } { 0.45 em } }
888         }
889         { \int_set:Nn \l_tmpa_int
890             { \dim_ratio:nn { \l_@@_l_dim - 0.6 em } { 0.45 em } }
891         }
892     }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

893     \dim_set:Nn \l_tmpa_dim
894     {
895         ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
896         \dim_ratio:nn { 0.45 em } \l_@@_l_dim
897     }
898     \dim_set:Nn \l_tmpb_dim
899     {
900         ( \g_@@_y_final_dim - \g_@@_y_initial_dim ) *
901         \dim_ratio:nn { 0.45 em } \l_@@_l_dim
902     }

```

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`.

```

903     \int_set:Nn \l_tmpb_int
904     {
905         \bool_if:NTF \l_@@_initial_open_bool
906         { \bool_if:NTF \l_@@_final_open_bool 1 0 }
907         { \bool_if:NTF \l_@@_final_open_bool 2 1 }
908     }

```

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.

```

909     \dim_gadd:Nn \g_@@_x_initial_dim
910     {
911         ( \g_@@_x_final_dim - \g_@@_x_initial_dim ) *
912         \dim_ratio:nn
913         { \l_@@_l_dim - 0.45 em * \l_tmpa_int } { \l_@@_l_dim * 2 } *
914         \l_tmpb_int
915     }

```

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

```

916     \dim_gadd:Nn \g_@@_y_initial_dim
917     {
918         ( \g_@@_y_final_dim - \g_@@_y_initial_dim ) *
919         \dim_ratio:nn
920         { \l_@@_l_dim - 0.45 em * \l_tmpa_int }
921         { \l_@@_l_dim * 2 } *
922         \l_tmpb_int
923     }
924     \begin { tikzpicture } [ overlay ]
925     \int_step_inline:nnnn 0 1 \l_tmpa_int
926     {
927         \pgfpathcircle
928         { \pgfpoint { \g_@@_x_initial_dim } { \g_@@_y_initial_dim } }
929         { 0.53 pt }
930         \pgfusepath { fill }
931         \dim_gadd:Nn \g_@@_x_initial_dim \l_tmpa_dim
932         \dim_gadd:Nn \g_@@_y_initial_dim \l_tmpb_dim
933     }

```

```

934         \end { tikzpicture }
935     }
936 }

```

12.7 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).

```

937 \cs_set_eq:NN \@@_ldots \ldots
938 \cs_set_eq:NN \@@_cdots \cdots
939 \cs_set_eq:NN \@@_vdots \vdots
940 \cs_set_eq:NN \@@_ddots \ddots
941 \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).

```

942 \cs_new_protected:Nn \@@_add_to_empty_cells:
943 {
944     \seq_gput_right:Nx \g_@@_empty_cells_seq
945     { \int_use:N \g_@@_row_int - \int_use:N \g_@@_column_int }
946 }

```

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}`).

```

947 \NewDocumentCommand \@@_Ldots { s }
948 {
949     \bool_if:nF { #1 } { \@@_instruction_of_type:n { Ldots } }
950     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_ldots }
951     \@@_add_to_empty_cells:
952 }

```

```

953 \NewDocumentCommand \@@_Cdots { s }
954 {
955     \bool_if:nF { #1 } { \@@_instruction_of_type:n { Cdots } }
956     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_cdots }
957     \@@_add_to_empty_cells:
958 }

```

```

959 \NewDocumentCommand \@@_Vdots { s }
960 {
961     \bool_if:nF { #1 } { \@@_instruction_of_type:n { Vdots } }
962     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_vdots }
963     \@@_add_to_empty_cells:
964 }

```

```

965 \NewDocumentCommand \@@_Ddots { s }
966 {
967     \bool_if:nF { #1 } { \@@_instruction_of_type:n { Ddots } }
968     \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_ddots }
969     \@@_add_to_empty_cells:
970 }

```

```

971 \NewDocumentCommand \@@_Iddots { s }
972 {
973   \bool_if:nF { #1 } { \@@_instruction_of_type:n { Iddots } }
974   \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_iddots }
975   \@@_add_to_empty_cells:
976 }

```

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

```

977 \cs_new_protected:Nn \@@_Hspace:
978 {
979   \@@_add_to_empty_cells:
980   \hspace
981 }

```

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

```

982 \cs_set_eq:NN \@@_old_multicolumn \multicolumn
983 \cs_new:Nn \@@_multicolumn:nnn
984 {
985   \@@_old_multicolumn { #1 } { #2 } { #3 }
986   \int_compare:nNnT #1 > 1
987   {
988     \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
989     { \int_eval:n \g_@@_row_int - \int_use:N \g_@@_column_int }
990     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
991   }
992   \int_gadd:Nn \g_@@_column_int { #1 - 1 }
993 }

```

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

```

994 \NewDocumentCommand \@@_Hdotsfor { 0 { } m }
995 {
996   \tl_gput_right:Nx \g_@@_lines_to_draw_tl
997   {
998     \exp_not:N \@@_draw_Hdotsfor:nnn
999     { \int_use:N \g_@@_row_int }
1000     { \int_use:N \g_@@_column_int }
1001     { #2 }
1002   }
1003   \prg_replicate:nn { #2 - 1 } { & }
1004 }

1005 \cs_new_protected:Nn \@@_draw_Hdotsfor:nnn
1006 {
1007   \bool_set_false:N \l_@@_initial_open_bool
1008   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

1009 \int_set:Nn \l_@@_initial_i_int { #1 }
1010 \int_set:Nn \l_@@_final_i_int { #1 }

```

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

```

1011 \int_compare:nNnTF #2 = 1
1012 {
1013   \int_set:Nn \l_@@_initial_j_int 1
1014   \bool_set_true:N \l_@@_initial_open_bool
1015 }
1016 {

```

```

1017 \int_set:Nn \l_tmpa_int { #2 - 1 }
1018 \@@_if_not_empty_cell:nnTF \l_@@_initial_i_int \l_tmpa_int
1019 { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
1020 {
1021 \int_set:Nn \l_@@_initial_j_int {#2}
1022 \bool_set_true:N \l_@@_initial_open_bool
1023 }
1024 }
1025 \int_compare:nNnTF { #2 + #3 - 1 } = \g_@@_column_int
1026 {
1027 \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
1028 \bool_set_true:N \l_@@_final_open_bool
1029 }
1030 {
1031 \int_set:Nn \l_tmpa_int { #2 + #3 }
1032 \@@_if_not_empty_cell:nnTF \l_@@_final_i_int \l_tmpa_int
1033 { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
1034 {
1035 \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
1036 \bool_set_true:N \l_@@_final_open_bool
1037 }
1038 }
1039 \bool_if:nT { \l_@@_initial_open_bool || \l_@@_final_open_bool }
1040 \@@_create_extra_nodes:
1041 \@@_actually_draw_ldots:
1042 }

```

12.8 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.

```

1043 \cs_new_protected:Nn \@@_line:nn
1044 {
1045 \dim_zero_new:N \g_@@_x_initial_dim
1046 \dim_zero_new:N \g_@@_y_initial_dim
1047 \dim_zero_new:N \g_@@_x_final_dim
1048 \dim_zero_new:N \g_@@_y_final_dim
1049 \bool_set_false:N \l_@@_initial_open_bool
1050 \bool_set_false:N \l_@@_final_open_bool
1051 \begin { tikzpicture }
1052 \path~( #1 ) ~---~( #2 ) ~node[at~start]~( i ) ~{}~node[at~end]~( f ) ~{} ;
1053 \tikz@parse@node \pgfutil@firstofone ( i )
1054 \dim_gset:Nn \g_@@_x_initial_dim \pgf@x
1055 \dim_gset:Nn \g_@@_y_initial_dim \pgf@y
1056 \tikz@parse@node \pgfutil@firstofone ( f )
1057 \dim_gset:Nn \g_@@_x_final_dim \pgf@x
1058 \dim_gset:Nn \g_@@_y_final_dim \pgf@y
1059 \end { tikzpicture }
1060 \@@_draw_tikz_line:
1061 }

```

The commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

12.9 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.

```

1062 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

1063 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
1064 {
1065   auto-columns-width .code:n =
1066   {
1067     \bool_set_true:N \l_@@_block_auto_columns_width_bool
1068     \dim_gzero_new:N \g_@@_max_cell_width_dim
1069     \bool_set_true:N \l_@@_auto_columns_width_bool
1070   }
1071 }

1072 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
1073 {
1074   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
1075   \int_zero_new:N \l_@@_first_env_block_int
1076   \int_set:Nn \l_@@_first_env_block_int { \g_@@_env_int + 1 }
1077 }

```

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).

```

1078 {
1079   \bool_if:NT \l_@@_block_auto_columns_width_bool
1080   {
1081     \iow_now:Nn \@mainaux \ExplSyntaxOn
1082     \int_step_inline:nnnn \l_@@_first_env_block_int 1 \g_@@_env_int
1083     {
1084       \iow_now:Nx \@mainaux
1085       {
1086         \cs_gset:cpn { @@ _ max _ cell _ width _ ##1 }
1087         { \dim_use:N \g_@@_max_cell_width_dim }
1088       }
1089     }
1090     \iow_now:Nn \@mainaux \ExplSyntaxOff
1091   }
1092 }

```

12.10 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}.

```

1093 \keys_define:nn { NiceMatrix / NiceArrayC }
1094 {
1095   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
1096   code-for-last-col .value_required:n = true ,
1097   unknown .code:n = \@@_error:n { Unknown~option~for~NiceArrayC }
1098 }

1099 \@@_msg_new:nnn { Unknown~option~for~NiceArrayC }
1100 {
1101   The~option~'\tl_use:N\l_keys_key_tl'~is~unknown~for~the~environment~
1102   \{\@currenvir\}. \\
1103   If~you~go~on,~it~will~be~ignored. \\
1104   For~a~list~of~the~available~options,~type~H~<return>.
1105 }
1106 {
1107   The~available~options~are~(in~alphabetic~order):~
1108   code~after,~
1109   code~for~last~col,~
1110   columns~width,~

```

```

1111     create-extra-nodes,~
1112     extra-left-margin,~
1113     extra-right-margin,~
1114     left-margin,~
1115     name,~
1116     nullify-dots,~
1117     parallelize-diags~
1118     renew-dots~
1119     and~right-margin.
1120 }

```

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}`.

```

1121 \cs_new_protected:Nn \@@_NiceArrayC:n
1122 {
1123     \bool_set_true:N \l_@@_exterior_column_bool
1124     \begin { NiceArray }

```

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

```

1125     { #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:.`

```

1126         >
1127         {
1128             \int_gincr:N \g_@@_column_int
1129             \int_gset:Nn \g_@@_column_total_int
1130             { \int_max:nn \g_@@_column_total_int \g_@@_column_int }
1131             \hbox_set:Nw \l_tmpa_box $ % $
1132             \l_@@_code_for_last_col_tl
1133         }
1134     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).

```

1135     < { $ % $
1136         \hbox_set_end:
1137         \dim_gset:Nn \g_@@_width_last_col_dim
1138         {
1139             \dim_max:nn
1140             \g_@@_width_last_col_dim
1141             { \box_wd:N \l_tmpa_box }
1142         }
1143         \skip_horizontal:n { - 2 \arraycolsep }

```

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

```

1144         \hbox_overlap_right:n
1145         {
1146             \skip_horizontal:n
1147             {
1148                 2 \arraycolsep +
1149                 \l_@@_right_margin_dim +
1150                 \l_@@_extra_right_margin_dim
1151             }
1152             \tikz
1153             [
1154                 remember-picture ,
1155                 inner~sep = \c_zero_dim ,

```

```

1156         minimum-width = \c_zero_dim ,
1157         baseline
1158     ]
1159     \node
1160     [
1161         anchor = base ,
1162         name =
1163             nm -
1164             \int_use:N \g_@@_env_int -
1165             \int_use:N \g_@@_row_int -
1166             \int_use:N \g_@@_column_int ,
1167         alias =
1168             \str_if_empty:NF \l_@@_name_str
1169             {
1170                 \l_@@_name_str -
1171                 \int_use:N \g_@@_row_int -
1172                 \int_use:N \g_@@_column_int
1173             }
1174     ]
1175     { \box_use:N \l_tmpa_box } ;
1176 }
1177 }
1178 }
1179 }

```

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.

```

1180 \NewDocumentEnvironment { NiceArrayCwithDelims } { m m O { } m ! O { } }
1181 {
1182     \dim_gzero_new:N \g_@@_width_last_col_dim
1183     \keys_set:nn { NiceMatrix / NiceArrayC } { #3 , #5 }
1184     \bool_set_false:N \l_@@_exterior_arraycolsep_bool
1185     \str_set:Nn \l_@@_pos_env_str c
1186     \left #1
1187     \@@_NiceArrayC:n { #4 }
1188 }
1189 {
1190     \end { NiceArray }
1191     \right #2
1192     \skip_horizontal:n \g_@@_width_last_col_dim
1193 }

```

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.

```

1194 \NewDocumentEnvironment { pNiceArrayC } { }
1195 { \NiceArrayCwithDelims ( ) }
1196 { \endNiceArrayCwithDelims }
1197 \NewDocumentEnvironment { vNiceArrayC } { }
1198 { \NiceArrayCwithDelims | | }
1199 { \endNiceArrayCwithDelims }
1200 \NewDocumentEnvironment { VNiceArrayC } { }
1201 { \NiceArrayCwithDelims \l \l }
1202 { \endNiceArrayCwithDelims }
1203 \NewDocumentEnvironment { bNiceArrayC } { }
1204 { \NiceArrayCwithDelims [ ] }
1205 { \endNiceArrayCwithDelims }
1206 \NewDocumentEnvironment { BNiceArrayC } { }
1207 { \NiceArrayCwithDelims \{ \} }
1208 { \endNiceArrayCwithDelims }

```

12.11 The environment {pNiceArrayRC}

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

```

1209 \keys_define:nn { NiceMatrix / NiceArrayRC }
1210 {
1211   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
1212   code-for-first-row .value_required:n = true ,
1213   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
1214   code-for-last-col .value_required:n = true ,
1215   unknown .code:n = \@@_error:n { Unknown-option-for-NiceArrayRC }
1216 }
1217 \@@_msg_new:nnn { Unknown-option-for-NiceArrayRC }
1218 {
1219   The~option~'\tl_use:N\l_keys_key_tl'~is-unknown~for~the~environment~
1220   \{ \@@_currenvir \}. \\\
1221   If~you~go~on,~it~will~be~ignored. \\\
1222   For~a~list~of~the~available~options,~type~H~<return>.
1223 }
1224 {
1225   The~available~options~are~(in~alphabetic~order):~
1226   code-after,~
1227   code-for-last-col,~
1228   code-for-first-row,~
1229   columns-width,~
1230   create-extra-nodes,~
1231   extra-left-margin,~
1232   extra-right-margin,~
1233   left-margin,~
1234   name,~
1235   nullify-dots,~
1236   parallelize-diags,~
1237   renew-dots~
1238   and~right-margin.
1239 }

```

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.

```

1240 \NewDocumentEnvironment { NiceArrayRCwithDelims } { m m O { } m ! O { } }
1241 {
1242   \int_zero:N \l_@@_nb_first_row_int
1243   \dim_gzero_new:N \g_@@_width_last_col_dim
1244   \keys_set:nn { NiceMatrix / NiceArrayRC } { #3 , #5 }
1245   \bool_set_false:N \l_@@_exterior_arraycolsep_bool
1246   \str_set:Nn \l_@@_pos_env_str c
1247   \box_clear_new:N \l_@@_the_array_box
1248   \hbox_set:Nw \l_@@_the_array_box
1249   $ % $
1250   \@@_NiceArrayC:n { #4 }
1251 }
1252 {
1253   \end { NiceArray }
1254   $ % $
1255   \hbox_set_end:
1256   \dim_set:Nn \l_tmpa_dim
1257   {
1258     (
1259       \dim_max:nn
1260       { 12 pt }
1261       { \g_@@_max_ht_row_one_dim + \g_@@_max_dp_row_zero_dim }
1262     )

```



```

1263     + \g_@@_max_ht_row_zero_dim - \g_@@_max_ht_row_one_dim
1264 }
1265 \hbox_set:Nn \l_tmpa_box
1266 {
1267     $ % $
1268     \left #1
1269     \vcenter
1270     {
1271         \skip_vertical:n { - \l_tmpa_dim }
1272         \box_use_drop:N \l_@@_the_array_box
1273     }
1274     \right #2
1275     $ % $
1276     \skip_horizontal:n \g_@@_width_last_col_dim
1277 }
1278 \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
1279 \box_use_drop:N \l_tmpa_box
1280 }

```

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.

```

1281 \NewDocumentEnvironment { pNiceArrayRC } { }
1282 { \NiceArrayRCwithDelims ( ) }
1283 { \endNiceArrayRCwithDelims }
1284 \NewDocumentEnvironment { bNiceArrayRC } { }
1285 { \NiceArrayRCwithDelims [ ] }
1286 { \endNiceArrayRCwithDelims }
1287 \NewDocumentEnvironment { vNiceArrayRC } { }
1288 { \NiceArrayRCwithDelims | | }
1289 { \endNiceArrayRCwithDelims }
1290 \NewDocumentEnvironment { VNiceArrayRC } { }
1291 { \NiceArrayRCwithDelims \l \l }
1292 { \endNiceArrayRCwithDelims }
1293 \NewDocumentEnvironment { BNiceArrayRC } { }
1294 { \NiceArrayRCwithDelims \{ \} }
1295 { \endNiceArrayRCwithDelims }

```

12.12 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

1296 \cs_generate_variant:Nn \dim_min:nn { v n }
1297 \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`.

```

1298 \cs_new_protected:Nn \@@_create_extra_nodes:
1299 {
1300     \begin { tikzpicture } [ remember-picture , overlay ]
1301         \int_step_variable:nnnNn \l_@@_nb_first_row_int 1 \g_@@_row_int \@@_i
1302         {
1303             \dim_zero_new:c { l_@@_row_\@@_i_min_dim }

```

```

1304     \dim_set_eq:cN { l_@@_row\_@@_i _min_dim } \c_max_dim
1305     \dim_zero_new:c { l_@@_row\_@@_i _max_dim }
1306     \dim_set:cn { l_@@_row\_@@_i _max_dim } { - \c_max_dim }
1307   }
1308   \int_step_variable:nnnNn 1 1 \g_@@_column_total_int \@@_j
1309   {
1310     \dim_zero_new:c { l_@@_column\_@@_j _min_dim }
1311     \dim_set_eq:cN { l_@@_column\_@@_j _min_dim } \c_max_dim
1312     \dim_zero_new:c { l_@@_column\_@@_j _max_dim }
1313     \dim_set:cn { l_@@_column\_@@_j _max_dim } { - \c_max_dim }
1314   }

```

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

```

1315   \int_step_variable:nnNn \l_@@_nb_first_row_int \g_@@_row_int \@@_i
1316   {
1317     \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.

```

1318       { \cs_if_exist:cT
1319         { 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`.

```

1320       {
1321         \tikz@parse@node \pgfutil@firstofone
1322         ( nm - \int_use:N \g_@@_env_int
1323           - \@@_i - \@@_j .south-west )
1324         \dim_set:cn { l_@@_row\_@@_i _min_dim }
1325         { \dim_min:vn { l_@@_row _ \@@_i _min_dim } \pgf@y }
1326         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i - \@@_j }
1327         {
1328           \dim_set:cn { l_@@_column _ \@@_j _min_dim }
1329           { \dim_min:vn { l_@@_column _ \@@_j _min_dim } \pgf@x }
1330         }

```

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`.

```

1331         \tikz@parse@node \pgfutil@firstofone
1332         ( nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j .north-east )
1333         \dim_set:cn { l_@@_row _ \@@_i _max_dim }
1334         { \dim_max:vn { l_@@_row _ \@@_i _max_dim } \pgf@y }
1335         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i - \@@_j }
1336         {
1337           \dim_set:cn { l_@@_column _ \@@_j _max_dim }
1338           { \dim_max:vn { l_@@_column _ \@@_j _max_dim } \pgf@x }
1339         }
1340       }
1341     }
1342   }

```

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`).

```

1343   \tikzset { name~suffix = -medium }
1344   \@@_create_nodes:

```

For “large nodes”, the eventual “first row” and “last column” (in environments like `{pNiceArrayRC}`) don't interfere. 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`).²¹

```

1345   \int_set:Nn \l_@@_nb_first_row_int 1

```

²¹We 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`.

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`.

```

1346 \int_step_variable:nnnNn 1 1 { \g_@@_row_int - 1 } \@@_i
1347 {
1348   \dim_set:cn { l_@@_row _ \@@_i _ min _ dim }
1349   {
1350     (
1351       \dim_use:c { l_@@_row _ \@@_i _ min _ dim } +
1352       \dim_use:c { l_@@_row _ \int_eval:n { \@@_i + 1 } _ max _ dim }
1353     )
1354     / 2
1355   }
1356   \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i + 1 } _ max _ dim }
1357   { l_@@_row _ \@@_i _ min _ dim }
1358 }
1359 \int_step_variable:nnnNn 1 1 { \g_@@_column_int - 1 } \@@_j
1360 {
1361   \dim_set:cn { l_@@_column _ \@@_j _ max _ dim }
1362   {
1363     (
1364       \dim_use:c
1365       { l_@@_column _ \@@_j _ max _ dim } +
1366       \dim_use:c
1367       { l_@@_column _ \int_eval:n { \@@_j + 1 } _ min _ dim }
1368     )
1369     / 2
1370   }
1371   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j + 1 } _ min _ dim }
1372   { l_@@_column _ \@@_j _ max _ dim }
1373 }
1374 \dim_sub:cn
1375 { l_@@_column _ 1 _ min _ dim }
1376 \g_@@_left_margin_dim
1377 \dim_add:cn
1378 { l_@@_column _ \int_use:N \g_@@_column_int _ max _ dim }
1379 \g_@@_right_margin_dim

```

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

```

1380 \tikzset { name~suffix = -large }
1381 \@@_create_nodes:
1382 \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.

```

1383 \cs_set:Nn \@@_create_extra_nodes: { }
1384 }

```

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.

```

1385 \cs_new_protected:Nn \@@_create_nodes:
1386 {
1387   \int_step_variable:nnnNn \l_@@_nb_first_row_int 1 \g_@@_row_int \@@_i
1388   {
1389     \int_step_variable:nnnNn 1 1 \g_@@_column_total_int \@@_j

```

We create two punctual 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.

```

1390 {
1391     \coordinate ( @@~south~west )
1392         at ( \dim_use:c { l_@@_column_ \@@_j _min_dim } ,
1393             \dim_use:c { l_@@_row_ \@@_i _min_dim } ) ;
1394     \coordinate ( @@~north~east )
1395         at ( \dim_use:c { l_@@_column_ \@@_j _max_dim } ,
1396             \dim_use:c { l_@@_row_ \@@_i _max_dim } ) ;

```

We can eventually draw the rectangular node for the cell ($\backslash@@_i$ - $\backslash@@_j$). 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`.

```

1397     \node
1398     [
1399         node~contents = { } ,
1400         fit = ( @@~south~west ) ( @@~north~east ) ,
1401         inner~sep = \c_zero_dim ,
1402         name = nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j ,
1403         alias =
1404             \str_if_empty:NF \g_@@_name_str
1405                 { \g_@@_name_str - \@@_i - \@@_j }
1406     ]
1407     ;
1408 }
1409 }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

1410 \@@_seq_mapthread_function:NNN
1411     \g_@@_multicolumn_cells_seq
1412     \g_@@_multicolumn_sizes_seq
1413     \@@_node_for_multicolumn:nn
1414 }
1415 \cs_new_protected:Npn \@@_extract_coords: #1 - #2 \q_stop
1416 {
1417     \cs_set:Npn \@@_i { #1 }
1418     \cs_set:Npn \@@_j { #2 }
1419 }

```

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”).

```

1420 \cs_new_protected:Nn \@@_node_for_multicolumn:nn
1421 {
1422     \@@_extract_coords: #1 \q_stop
1423     \coordinate ( @@~south~west ) at
1424     (
1425         \dim_use:c { l_@@_column_ \@@_j _min_dim } ,
1426         \dim_use:c { l_@@_row_ \@@_i _min_dim }
1427     ) ;
1428     \coordinate ( @@~north~east ) at
1429     (
1430         \dim_use:c { l_@@_column_ \int_eval:n { \@@_j + #2 - 1 } _max_dim } ,
1431         \dim_use:c { l_@@_row_ \@@_i _max_dim }
1432     ) ;
1433     \node
1434     [
1435         node~contents = { } ,
1436         fit = ( @@~south~west ) ( @@~north~east ) ,
1437         inner~sep = \c_zero_dim ,
1438         name = nm - \int_use:N \g_@@_env_int - \@@_i - \@@_j ,
1439         alias =
1440             \str_if_empty:NF \g_@@_name_str { \g_@@_name_str - \@@_i - \@@_j }
1441     ]

```

```

1442     ;
1443 }

```

12.13 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.

```

1444 \ProcessKeysOptions {NiceMatrix}

```

12.14 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`.

```

1445 \cs_new:Npn \@@_seq_mapthread_function:NNN #1 #2 #3
1446 {
1447   \group_begin:

```

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

```

1448   \int_step_inline:nnnn 1 1 { \seq_count:N #1 }
1449   {
1450     \seq_pop:NN #1 \l_tmpa_tl
1451     \seq_pop:NN #2 \l_tmpb_tl
1452     \exp_args:NVV #3 \l_tmpa_tl \l_tmpb_tl
1453   }
1454   \group_end:
1455 }

```

```

1456 \cs_set_protected:Npn \@@_renew_matrix:
1457 {
1458   \RenewDocumentEnvironment { pmatrix } { } { }
1459   { \pNiceMatrix }
1460   { \endpNiceMatrix }
1461   \RenewDocumentEnvironment { vmatrix } { } { }
1462   { \vNiceMatrix }
1463   { \endvNiceMatrix }
1464   \RenewDocumentEnvironment { Vmatrix } { } { }
1465   { \VNiceMatrix }
1466   { \endVNiceMatrix }
1467   \RenewDocumentEnvironment { bmatrix } { } { }
1468   { \bNiceMatrix }
1469   { \endbNiceMatrix }
1470   \RenewDocumentEnvironment { Bmatrix } { } { }
1471   { \BNiceMatrix }
1472   { \endBNiceMatrix }
1473 }

```

13 History

13.1 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.

13.2 Changes between versions 1.1 and 1.2

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

13.3 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).

13.4 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.

13.5 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.

13.6 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”.

13.7 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 StackExchange²², Tikz externalization is now deactivated in the environments of the extension `nicematrix`.²³

13.8 Changes between version 2.1 and 2.1.2

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

13.9 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`).

$$\left(\begin{array}{ccc} & C_j & \\ 0 & \vdots & 0 \\ & a \cdots \cdots & \\ 0 & & 0 \end{array} \right) L_i$$

²²cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

²³Before 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`.

13.10 Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See <https://www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end>

13.11 Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.