The package nicematrix*

F. Pantigny
fpantigny@wanadoo.fr

July 6, 2018

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, mathtools and tikz.

This package provides some new tools to draw mathematical matrices. The main features are the following:
- continuous dotted lines;
- a first row and a last column for labels;
- a control of the width of the columns.

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

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

An example for the continuous dotted lines

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

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

This code composes the matrix A on the right.

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

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 2.0 of nicematrix, at the date of 2018/07/06.

2 The environments of this extension

The extension nicematrix defines the following new environments.

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

By default, the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` behave almost exactly as the corresponding environments of amsmath (and mathtools): `{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. 7 the section relating to `{pNiceArrayC}`.

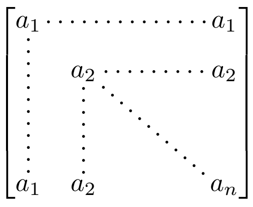
The environments with `RC`, `{pNiceArrayRC}`, `{bNiceArrayRC}`, `{BNiceArrayRC}`, `{vNiceArrayRC}`, `{VNiceArrayRC}` are similar to the environment `{NiceArray}` but create an exterior row (above the main matrix) and an exterior column. See p. 8 the section relating to `{pNiceArrayRC}`.

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`.[1]

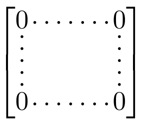
Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells[2] on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones.

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



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

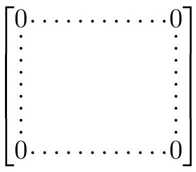
```
\begin{bNiceMatrix}
0      & \Cdots & 0      \\
\Vdots &        & \Vdots \\
0      & \Cdots & 0
\end{bNiceMatrix}
```



[1] The command `\iddots`, defined in nicematrix, is a variant of `\ddots` with dots going forward: $\iddots$. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of unicode-math.
[2] The precise definition of a "non-empty cell" is given below.

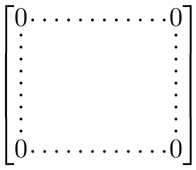
However, one may want a larger matrix. Usually, in such a case, the users of LaTeX add a new row and a new column. It's possible to use the same method with nicematrix:

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



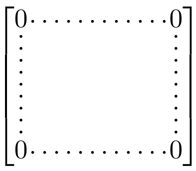
In the first column of this exemple, there are two instructions `\Vdots` but only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF).
However, useless computations are performed by TeX before detecting that both instructions would eventually yield the same dotted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

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



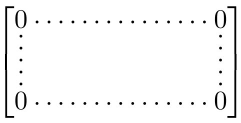
In fact, in this example, it would be possible to draw the same matrix without starred commands with the following code:

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



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.[3]
However, a command `\hspace*` might interfer with the construction of the dotted lines. That's why the package nicematrix provides a command `\Hspace` which is a variant of `\hspace` transparent for the dotted lines of nicematrix.

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



3.1 The option nullify-dots

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

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

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

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

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

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

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

By default, with nicematrix, if we replace `{pmatrix}` by `{pNiceMatrix}` and `\vdots` by `\Vdots` (or `\Vdots*` for efficiency), the geometry of the matrix is not changed.

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

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

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

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

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

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

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, they should use instead the command `\Hdotsfor` and they will have the same style of dotted lines. We recommand not to use this command in new code since it's more coherent to use exclusively `\Cdots`, `\Vdots`, `\Ddots`, etc. However, it can be useful in adapting existing code, especially when using the option `transparent` (see below).

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

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

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

3.3 How to generate the continuous dotted lines transparently

The package nicematrix provides an option called `transparent` for using existing code transparently in the environments `{matrix}`. This option can be set as option of `\usepackage` or with the command `\NiceMatrixOptions`.
In fact, this option is an alias for the conjonction of two options: `renew-dots` and `renew-matrix`.

- The option `renew-dots`

  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`[4] 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` — and `mathtools`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

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

Therefore, with the option `transparent`, a classical code gives directly the ouput of nicematrix.

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

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

4 The width of the columns

In the environments with an explicit preamble (like `{NiceArray}`, `{pNiceArrayC}`, 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.**

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

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

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

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

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

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`.[5]

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

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

5 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 matrix (with a key called `name`). Then, the nodes are accessible through the names "*name-i-j*" where *name* is the name given to the matrix and i and j the number 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`.

6 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).
Morevover, a special command, called `\line` is available to draw directly dotted lines between nodes.

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

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

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

7 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`[6] 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.

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

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

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

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

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

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

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

8 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 column `L` column.
A special option, called `code-for-last-col`, specifies tokens that will be inserted before each cell of the last column. The option `columns-width` doesn't apply to this external column.

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

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

```
       &    &    &\Ddots &    &         & \Vdots & \Vdots \\
\Vdots &    &    &\Ddots &    & 0       & \\
0      &    &    &\Cdots & 0 & 1       & 0      & L_n \gets L_n-L_1
\end{pNiceArrayC}$
```

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

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

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

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

9 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).
For technical reasons, the key `columns-width` is mandatory, with an numerical value (e.g. 1 cm) or with the special value `auto`. This implies that all the columns of an environment `{pNiceArrayRC}` have necessarily the same width.
A special option, called `code-for-first-row`, specifies tokens that will be inserted before each cell of the first row.

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

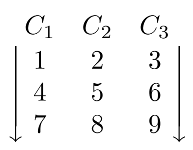
$$\begin{array}{cl} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} & \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} & \begin{array}{l} L_1 \\ L_2 \\ L_3 \end{array} \end{array}$$

For technical reasons, the first line of the array (which will be composed "above" the main matrix) must *not* contain the command `\\` and no verbatim material.

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`).

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 delimiter used in the construction of the matrix. It's possible to use `{NiceArrayRCwithDelims}` if we want to use atypical delimiters.

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



10 Technical remarks

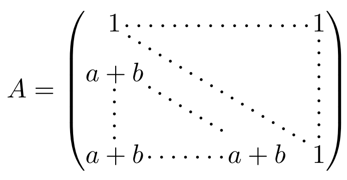
10.1 Diagonal lines

By default, all the diagonal lines[7] of a same array are "parallelized". That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

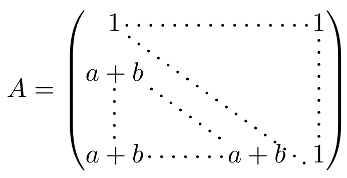
In the following examples, the first `\Ddots` instruction is written in color:

Example with parallelization (default):

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

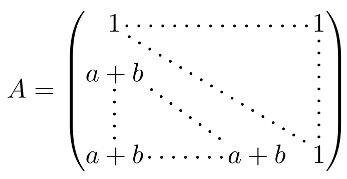


```
$A = \begin{pNiceMatrix}
1      & \Cdots &        & 1      \\
a+b    &        &        & \Vdots \\
\Vdots & \Ddots & \Ddots &        \\
a+b    & \Cdots & a+b    & 1
\end{pNiceMatrix}$
```



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

The same example without parallelization:



10.2 The "empty" cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides[8]. 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:

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

[8]If nicematrix can't find these cells, an error `Imposible instruction` is raised. Nevertheless, with the option `silent`, these instructions are discarded silently.

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

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

- Each cell whose TeX ouput has a width less than 0.5 pt is empty.
- A cell which contains a command `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` or `\Iddots` and their starred versions is empty. We recall that theses 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.

A dotted line must be delimited by two non-empty cells. If it's not possible to find one of these cells whitin the boundaries of the matrix, an error is issued and the instruction is ignored.

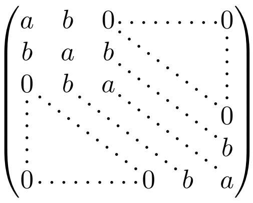
10.3 The option exterior-arraycolsep

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.[9]
The environment `{matrix}` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of amsmath and mathtools 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`. With this option, `{NiceArray}` will insert the same horizontal spaces as the environment `{array}`. This option can be set for a given environment or globally with the command `\NiceMatrixOptions`.

11 Examples

A tridiagonal matrix:

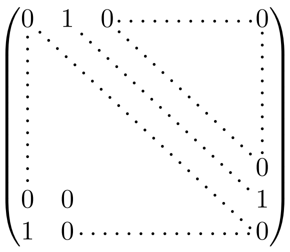
```
$\begin{pNiceMatrix}[nullify-dots]
a      & b      & 0      &        & \Cdots & 0      \\
b      & a      & b      & \Ddots &        & \Vdots \\
0      & b      & a      & \Ddots &        &        \\
       & \Ddots & \Ddots & \Ddots &        & 0      \\
\Vdots &        &        &        &      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



A permutation matrix:

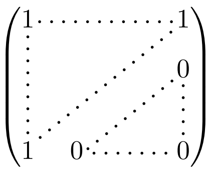
[9] In the documentation of `{amsmath}`, we can read: *The extra space of* `\arraycolsep` *that array adds on each side is a waste so we remove it [in* `{matrix}`*] (perhaps we should instead remove it from array in general, but that's a harder task).* It's possible to suppress these spaces for a given environment `{array}` with a construction like `\begin{array}{@{}ccccc@{}}`.

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



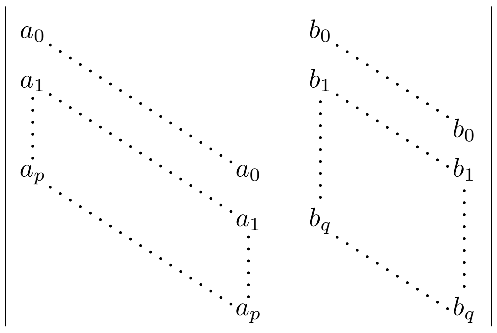
An example with `\Iddots`:

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



An example for the resultant of two polynoms :

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



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

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

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

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

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

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

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

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

In the following example, we want to highlight a row of the matrix. To do so, we create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library `fit`. This Tikz node is filled after the construction of the matrix. In order to see the text *under* this node, we have to use transparency with the `blend mode` equal to `multiply`.

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

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

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

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
```

12 Implementation

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

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

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

```
\RequirePackage{l3keys2e}
\ProvidesExplPackage
  {nicematrix}
  {\myfiledate}
  {\myfileversion}
  {Draws nice dotted lines in matrix environments}
```

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

We load array, mathtools (mathtools may be considered as the successor of amsmath) and tikz.

```
\RequirePackage{array}
\RequirePackage{mathtools}
\RequirePackage{tikz}
```

The package xparse will be used to define the environment `{NiceMatrix}`, its variants and the document-level commands (`\NiceMatrixOptions`, etc.).

```
\RequirePackage{xparse}
```

12.2 Technical definitions

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

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

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

In the environment `{NiceMatrix}`, the command `\multicolumn` will be linked to the following command `\@@_multicolumn:nn` but only if the option `renew-matrix` is not set. Indeed, if the option `renew-matrix` is used, we want to let the possibility to the user to use `\multicolumn` in some matrices without dotted lines and to have the automatic dotted lines of nicematrix in other matrices.

```
\cs_new_protected:Nn \@@_multicolumn:nn
       {\msg_error:nn {nicematrix} {Multicolumn~forbidden}}
```

This command `\@@_multicolumn:nn` takes two arguments, and therefore, the first two arguments of `\column` will be gobbled.

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

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

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

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

The sequence `\g_@@_names_seq` will be the list of all the names of environments used (via the option `name`) in the document : two environments must not have the same name.

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

The integer `\l_@@_number_of_first_row_int` is the number of the first row of the array. With a value of 0 (the default), the first row of the array will have the number 1. Within the environment `{NiceArrayRC}`, we will give the value -1 to `\l_@@_number_of_first_row_int` and, therefore, the first row of the array will have the number 0.

```
\int_new:N \l_@@_number_of_first_row_int
```

The flag `\l_@@_direct_NiceArray_bool` will indicate if the current environment `{NiceArray}` is "direct" : that means that it is not encompassed in another environment of the extension nicematrix like `{NiceMatrix}` or `{pNiceArrayC}`.

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

The token list `\l_@@_code_for_cell_tl` is a code that will be inserted in the beginning of each cell of the array (before the symbol `$` of the mathematical mode).

```
\tl_new:N \l_@@_code_for_cell_tl
```

12.3 The options

The token list `\l_@@_pos_env_tl` 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 `{NiceMatrix}` (and its variants) and `{pNiceArrayC}` (and its variants), the value will programmatically be fixed to `c`. For the environment `{NiceArray}`, however, the three values `t`, `c` and `b` are possible.

```
\tl_new:N \l_@@_pos_env_tl
\tl_set:Nn \l_@@_pos_env_tl c
```

The boolean `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (but neither for `{NiceMatrix}`, `{pNiceArrayC}`, `{pNiceArrayRC}` and their variants even if these environments rely upon `{NiceArray}`).

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

The flag `\l_@@_parallelize_diags_bool` controls wether the diagonals are parallelized. The default is `true`.

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

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

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

The flag `\l_@@_renew_matrix_bool` will be raised if the option `renew-matrix` is used.

```
\bool_new:N \l_@@_renew_matrix_bool
```

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

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

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

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

The token list `\g_@@_name_tl` 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. This variable is global for technical reasons (some commands will be executed in an `\aftergroup` because we don't want to patch the commands of the package array).

```
\tl_new:N \g_@@_name_tl
```

We define a set of options which will be used with the command `NiceMatrixOptions`.[10]

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

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

```
      renew-dots        .bool_set:N = \l_@@_renew_dots_bool,
      renew-dots        .default:n  = true,
      RenewDots         .meta:n = renew-dots,
```

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

```
      renew-matrix      .code:n     = {\cs_set_eq:NN \env@matrix \NiceMatrix
                                       \bool_set_true:N \l_@@_renew_matrix_bool},
      renew-matrix      .value_forbidden:n = true,
      RenewMatrix       .meta:n     = renew-matrix,
      transparent       .meta:n     = {renew-dots,renew-matrix},
      transparent       .value_forbidden:n = true,
      Transparent       .meta:n     = transparent,
```

[10]Before the version 1.3, the names of the options were in "caml style" (like `ParallelizeDiagonals`) which was not a good idea. In version 1.4, the names are converted in lowercase with hyphens (like `parallelize-diags`). For compatibility, the old names are conversed.

Without the option `nullify-dots`, 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.). This option is set by default.

```
      nullify-dots      .bool_set:N = \l_@@_nullify_dots_bool ,
      nullify-dots      .default:n  = true,
      NullifyDots       .meta:n     = nullify-dots,
```

With the option `silent`, no error is generated for the impossible instructions. This option can be useful when adapting an existing code.

```
      silent            .code:n  = {\msg_redirect_name:nnn {nicematrix}
                                                           {Impossible~instruction}
                                                           {none}} ,
      silent            .value_forbidden:n = true,
      Silent            .meta:n            = silent,
```

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.[11]

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

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

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

If the option `columns-width` is used, all the columns will have the same width. In `\NiceMatrixOptions`, the special value `auto` is not available.

```
      columns-width     .code:n      = \str_if_eq:nnTF {#1} {auto}
                                          {\msg_error:nn {nicematrix}
                                                         {Option~auto~for~columns-width}}
                                          {\dim_set:Nn \l_@@_columns_width_dim {#1}},


      unknown .code:n  = \msg_error:nn {nicematrix} {Unknown~key~for~NiceMatrixOptions}}
```

```
\msg_new:nnnn {nicematrix}
              {Unknown~key~for~NiceMatrixOptions}
              {The~key~"\tl_use:N\l_keys_key_tl"~is~unknown~for~the~command
               \token_to_str:N \NiceMatrixOptions.\\
               If~you~go~on,~it~will~be~ignored.\\
               For~a~list~of~the~available~keys,~type~H~<return>.}
              {The~available~keys~are~(in~alphabetic~order):~
               code-for-last-col,~
               exterior-arraycolsep,~
               nullify-dots,~
               parallelize-diags,~
               renew-dots,~
               renew-matrix~
               and~transparent}
```

```
\msg_new:nnn {nicematrix}
             {Option~auto~for~columns-width}
             {You~can't~give~the~value~"auto"~to~the~option~"columns-width"~here.~
              If~you~go~on,~the~option~will~be~ignored.}
```

`\NiceMatrixOptions` is the command of the nicematrix package to fix options at the document level. The scope of these specifications is the current TeX group.

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

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

```
\keys_define:nn {NiceMatrix/NiceMatrix}
     {parallelize-diags .bool_set:N  = \l_@@_parallelize_diags_bool,
      parallelize-diags .default:n   = true,
      renew-dots        .bool_set:N  = \l_@@_renew_dots_bool,
      renew-dots        .default:n   = true,
      nullify-dots      .bool_set:N  = \l_@@_nullify_dots_bool ,
      nullify-dots      .default:n   = true,
```

The option `columns-width` is the width of the columns it we want the same width for all the columns of the array. A value of 0 pt means that the width of the column will be the natural width of the column.

```
      columns-width     .code:n      = \str_if_eq:nnTF {#1} {auto}
                                             {\bool_set_true:N
                                                  \l_@@_auto_columns_width_bool}
                                             {\dim_set:Nn \l_@@_columns_width_dim {#1}},
      name    .code:n                = {\seq_if_in:NnTF \g_@@_names_seq {#1}
                                             {\msg_error:nnn {nicematrix}
                                                             {Duplicate~name}
                                                             {#1}}
                                             {\seq_gput_left:Nn \g_@@_names_seq {#1}}
                                          \tl_gset:Nn \g_@@_name_tl {#1}},
      name    .value_required:n      = true,
```

Since we don't want to modify or patch any existing code, the "code after" will be inserted with an `\aftergroup`. Thus, we can't use a local variable for the "code after" and that's why we use a global variable.

```
      code-after         .tl_gset:N    = \g_@@_code_after_tl,
      code-after         .initial:n    = \c_empty_tl,
      code-after         .value_required:n = true,
      unknown .code:n  = \msg_error:nn {nicematrix} {Unknown~option~for~NiceMatrix}}
```

```
\msg_new:nnnn {nicematrix}
              {Unknown~option~for~NiceMatrix}
              {The~option~"\tl_use:N\l_keys_key_tl"~is~unknown~for~the~environment~
               \{NiceMatrix\}~and~its~variants.\\
               If~you~go~on,~it~will~be~ignored.\\
               For~a~list~of~the~available~options,~type~H~<return>.}
              {The~available~options~are~(in~alphabetic~order):~
               code-after,~
               columns-width,~
               name,~
               nullify-dots,~
               parallelize-diags~
               and~renew-dots.}
```

```
\keys_define:nn {NiceMatrix/NiceArray}
     {parallelize-diags    .bool_set:N = \l_@@_parallelize_diags_bool,
      parallelize-diags    .default:n  = true,
      renew-dots           .bool_set:N = \l_@@_renew_dots_bool,
      renew-dots           .default:n  = true,
      nullify-dots         .bool_set:N = \l_@@_nullify_dots_bool ,
      nullify-dots         .default:n  = true,
      exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
      exterior-arraycolsep .default:n  = true,
      columns-width        .code:n     = \str_if_eq:nnTF {#1} {auto}
                                             {\bool_set_true:N \l_@@_auto_columns_width_bool}
                                             {\dim_set:Nn \l_@@_columns_width_dim {#1}},
      columns-width        .value_required:n = true,
      name                 .code:n     = {\seq_if_in:NnTF \g_@@_names_seq {#1}
                                              {\msg_error:nnn {nicematrix}
                                                              {Duplicate~name}
                                                              {#1}}
                                              {\seq_gput_left:Nn \g_@@_names_seq {#1}}
                                            \tl_gset:Nn \g_@@_name_tl {#1}},
    name              .value_required:n = true,
```

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

```
    c                 .code:n    = \tl_set:Nn \l_@@_pos_env_tl c,
    t                 .code:n    = \tl_set:Nn \l_@@_pos_env_tl t,
    b                 .code:n    = \tl_set:Nn \l_@@_pos_env_tl b,
    code-after        .tl_gset:N = \g_@@_code_after_tl,
    code-after        .initial:n = \c_empty_tl,
    code-after        .value_required:n = true,
    unknown .code:n  = \msg_error:nn {nicematrix} {Unknown~option~for~NiceArray}}
```

```
\msg_new:nnnn {nicematrix}
              {Unknown~option~for~NiceArray}
              {The~option~"\tl_use:N\l_keys_key_tl"~is~unknown~for~the~environment~
               \{NiceArray\}.\\
               If~you~go~on,~it~will~be~ignored.\\
               For~a~list~of~the~available~options,~type~H~<return>.}
              {The~available~options~are~(in~alphabetic~order):~
               b,~
               c,~
               code-after,~
               columns-width,~
               exterior-arraycolsep,~
               name,~
               nullify-dots,~
               parallelize-diags,~
               renew-dots~
               and~t.}
```

12.4 The environments {NiceArray} and {NiceMatrix}

The pseudo-environment `\@@_Cell:n`–`\@@_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}`).
The argument of `\@@_Cell:n` is a letter `l`, `c` or `r` that describes the type of column: we store this letter at the end of `\g_@@_preamble_aux_tl` (see the redefinition of `\ialign` in the environment `{NiceArray}` for explanations).

```
\cs_new_protected:Nn \@@_Cell:n
   {\tl_gput_right:Nn \g_@@_preamble_aux_tl {#1}
```

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

```
    \int_gincr:N \g_@@_column_int
    \hbox_set:Nw \l_tmpa_box $ % $
          \l_@@_code_for_cell_tl }
```

```
\cs_new_protected:Nn \@@_end_Cell:
   {$ % $
    \hbox_set_end:
```

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

```
    \dim_gset:Nn \g_@@_max_cell_width_dim
          {\dim_max:nn \g_@@_max_cell_width_dim {\box_wd:N \l_tmpa_box}}
   \tl_if_empty:NTF \g_@@_name_tl
    {\tikz[remember~picture, inner~sep = 0pt, minimum~width = 0pt, baseline]
       \node [anchor = base,
              name = nm-\int_use:N \g_@@_env_int-
                        \int_use:N \g_@@_row_int-
                        \int_use:N \g_@@_column_int] }
    {\tikz[remember~picture, inner~sep = 0pt, minimum~width = 0pt, baseline]
       \node [anchor = base,
              name = nm-\int_use:N \g_@@_env_int-
```

```
                        \int_use:N \g_@@_row_int-
                        \int_use:N \g_@@_column_int,
              alias = \tl_use:N \g_@@_name_tl-
                       \int_use:N \g_@@_row_int-
                       \int_use:N \g_@@_column_int] }
      \bgroup
      \box_use:N \l_tmpa_box
      \egroup ;}
```

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

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

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

Here's the definition of `{NiceMatrix}`:

```
\NewDocumentEnvironment {NiceMatrix} {O{}}
    {\bool_set_false:N \l_@@_direct_NiceArray_bool
     \tl_gclear:N \g_@@_code_after_tl
     \keys_set:nn {NiceMatrix/NiceMatrix} {#1}
     \tl_set:Nn \l_@@_pos_env_tl c
     \bool_set_false:N \l_@@_exterior_arraycolsep_bool
         \NiceArray{*\c@MaxMatrixCols{C}}
      }
    {\endarray
     \skip_horizontal:n {-\arraycolsep}}
```

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

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

```
\cs_generate_variant:Nn \dim_set:Nn {Nx}
```

First, we test if we are yet in an environment `{NiceArray}` (nested environment are forbidden). It's easy to test whether we are in an environment `{NiceArray}` : a special command `\@@_in_NiceArray:` is defined.

```
\NewDocumentEnvironment {NiceArray} {O{} m O{}}
    {\cs_if_exist:NT \@@_in_NiceArray:
           {\msg_error:nn {nicematrix} {We~are~yet~in~an~environment~NiceArray}}
     \cs_set:Npn \@@_in_NiceArray: {--Void--}
     \aftergroup \@@_draw_lines:
```

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

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

Maybe, an encompassing environment {NiceMatrix} or {pNiceArrayC} (for example) has already set the value for `\g_@@_code_after_tl` (via the option `code-after`) and we don't want to erase this value. Thanks to the flag `\l_@@_direct_NiceArray_bool`, it's possible.

```
    \bool_if:NT \l_@@_direct_NiceArray_bool
            {\tl_gclear:N \g_@@_code_after_tl}
    \keys_set:nn {NiceMatrix/NiceArray} {#1,#3}
```

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

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

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

```
         \tl_if_empty:NF \g_@@_name_tl
           {\cs_if_free:cF {_@@_max_cell_width_\g_@@_name_tl}
              {\dim_set:Nx \l_@@_columns_width_dim
                   {\use:c {_@@_max_cell_width_\g_@@_name_tl}}}}
        }
```

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 (in particular to increment `\g_@@_row_int`). 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.[12]

```
    \cs_set:Npn \ialign
         {\everycr{\noalign{\int_gincr:N \g_@@_row_int
                              \int_gzero:N \g_@@_column_int
```

We want to have the preamble of the array in `\g_@@_preamble_tl` at the end of the array. However, some rows of the array may be shorter (that is to say without all the ampersands) and it's not possible to know which row will be the longest. That's why, during the construction of a row, we retrieve the corresponding preamble in `\g_@@_preamble_aux_tl`. At the end of the array, we are sure that the longest value will be the real preamble of the array (stored in `\g_@@_preamble_tl`).

```
                              \int_compare:nNnT {\tl_count:N \g_@@_preamble_aux_tl}
                                             > {\tl_count:N \g_@@_preamble_tl}
                                 {\tl_gset_eq:NN \g_@@_preamble_tl \g_@@_preamble_aux_tl}
                              \tl_gclear:N \g_@@_preamble_aux_tl}}
         \skip_zero:N \tabskip
         \cs_set:Npn \ialign {\everycr{}
                                \skip_zero:N \tabskip
                                \halign}
         \halign}
```

We define the new column types L, C and R that must be used instead of l, c and r in the preamble of {NiceArray}.

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

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

```
       {\newcolumntype{L}{wl{\dim_use:N \l_@@_columns_width_dim}}
```

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

```
        \newcolumntype{C}{wc{\dim_use:N \l_@@_columns_width_dim}}
        \newcolumntype{R}{wr{\dim_use:N \l_@@_columns_width_dim}}}
```

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.

```
    \cs_set_eq:NN \NC@find@w \relax
    \cs_set_eq:NN \NC@find@W \relax
```

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

```
    \newcolumntype{w}[2]
      {>{\hbox_set:Nw \l_tmpa_box
         \@@_Cell:n ##1}
       c
       <{\@@_end_Cell:
         \hbox_set_end:
         \makebox[##2][##1]{\box_use:N \l_tmpa_box}}}
    \newcolumntype{W}[2]
      {>{\hbox_set:Nw \l_tmpa_box
         \@@_Cell:n ##1}
       c
       <{\@@_end_Cell:
         \hbox_set_end:
         \cs_set_eq:NN \hss \hfil
         \makebox[##2][##1]{\box_use:N \l_tmpa_box}}}
```

The commands `\Ldots`, `\Cdots`, etc. will be defined only in the environment `{NiceArray}`.

```
    \cs_set_eq:NN \Ldots \@@_Ldots
    \cs_set_eq:NN \Cdots \@@_Cdots
    \cs_set_eq:NN \Vdots \@@_Vdots
    \cs_set_eq:NN \Ddots \@@_Ddots
    \cs_set_eq:NN \Iddots \@@_Iddots
    \cs_set_eq:NN \Hspace \@@_Hspace:
    \cs_set_eq:NN \NiceMatrixEndPoint \@@_NiceMatrixEndPoint:
    \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor
    \bool_if:NT \l_@@_renew_dots_bool
       {\cs_set_eq:NN \ldots \@@_Ldots
        \cs_set_eq:NN \cdots \@@_Cdots
        \cs_set_eq:NN \vdots \@@_Vdots
        \cs_set_eq:NN \ddots \@@_Ddots
        \cs_set_eq:NN \iddots \@@_Iddots
        \cs_set_eq:NN \hdotsfor \@@_Hdotsfor}
    \bool_if:NF \l_@@_renew_matrix_bool
      {\cs_set_eq:NN \multicolumn \@@_multicolumn:nn}
```

We have to remind the types of the columns (`l`, `c` or `r`) because we will use this information when we will draw the vertical dotted lines. That's why we store the types of the columns in `\g_@@_preamble_tl` (for example, if the preamble of `{NiceArray}` is `L*4{C}R`, the final value of `\g_@@_preamble_tl` will be `lccccr` if the user really uses the six columns).

```
    \tl_clear_new:N \g_@@_preamble_tl
    \tl_clear_new:N \g_@@_preamble_aux_tl
```

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

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

The counter `\g_@@_instruction_int` will count the instructions (`\Cdots`, `\Vdots`, `\Ddots`, etc.) in the matrix.

```
    \int_gzero_new:N \g_@@_instruction_int
```

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

```
    \int_gzero_new:N \g_@@_row_int
    \int_gset_eq:NN \g_@@_row_int \l_@@_number_of_first_row_int
```

The counter \g_@@_column_int will be used to count the columns of the array (it will be set to zero in \everycr). This counter is updated in the command \@@_Cell:n executed at the beginning of each cell. At the end of the array (in the beginning of \@@_draw_lines:), it will be set to the total number of columns of the array.

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

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

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

Eventually, the environment {NiceArray} is defined upon the environment {array}. The token list l_@@_pos_tl will contain one of the values t, c or b.

```
        \array[\l_@@_pos_env_tl]{#2}}
```

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

We create the variants of the environment {NiceMatrix}.

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

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

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

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

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

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

```
\prg_set_conditional:Npnn \@@_if_not_empty_cell:nn #1#2 {T}
```

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 wether we are in this situation considering the correspondant Tikz node.

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

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

```
         {\seq_if_in:NxTF \g_@@_empty_cells_seq
                          {\int_use:N #1-\int_use:N #2}
            {\prg_return_false:}
```

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

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

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

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

For each drawing instruction in the matrix (like `\Cdots`, etc.), we create a global property list to store the informations corresponding to the instruction. Such an property list will have three fields:

- a field "type" with the type of the instruction (`cdots`, `vdots`, `ddots`, etc.);
- a field "row" with the number of the row of the matrix where the instruction appeared;
- a field "column" with the number of the column of the matrix where the instruction appeared.

The argument of the following command `\@@_instruction_of_type:n` is the type of the instruction (`cdots`, `vdots`, `ddots`, etc.). This command creates the corresponding property list.

```
\cs_new_protected:Nn \@@_instruction_of_type:n
```

First, we increment the counter of the instructions (this counter is initialized in the beginning of the environment `{NiceMatrix}`). This incrementation is global because the command will be used in the cell of a `\halign`).

```
     {\int_gincr:N \g_@@_instruction_int
      \prop_put:Nnn \l_tmpa_prop {type} {#1}
      \prop_put:NnV \l_tmpa_prop {row} \g_@@_row_int
      \prop_put:NnV \l_tmpa_prop {column} \g_@@_column_int
```

The property list has been created in a local variable for convenience. Now, it will be stored in a global variable indicating the number of the instruction.

```
        \prop_gclear_new:c
           {g_@@_instruction_\int_use:N\g_@@_instruction_int _prop}
        \prop_gset_eq:cN
           {g_@@_instruction_\int_use:N\g_@@_instruction_int _prop}
           \l_tmpa_prop
        }
```

12.5 We draw the lines in the matrix

```
\cs_new_protected:Nn \@@_draw_lines:
    {
```

The last `\cr` in the array has incremented the counter `\g_@@_row_int`. That's why we decrement it to have the real number of rows.

```
       \int_decr:N \g_@@_row_int
```

The counter `\g_@@_column_int` will now be the total number of columns in the array.

```
       \int_set:Nn \g_@@_column_int {\tl_count:N \g_@@_preamble_tl}
```

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

```
       \seq_clear_new:N \l_@@_yet_drawn_seq
```

The following variables will be used further.

```
       \int_zero_new:N \l_@@_type_int
       \int_zero_new:N \l_@@_row_int
       \int_zero_new:N \l_@@_column_int
       \int_zero_new:N \l_@@_di_int
       \int_zero_new:N \l_@@_dj_int
```

By befault, the diagonal lines will be parallelized[13]. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to known wether a diagonal is the first of its type in the current `{NiceArray}` environment.

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

The dimensions `\l_@@_delta_x_one_dim` and `\l_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\l_@@_delta_x_two_dim` and `\l_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```
             \dim_zero_new:N \l_@@_delta_x_one_dim
             \dim_zero_new:N \l_@@_delta_y_one_dim
             \dim_zero_new:N \l_@@_delta_x_two_dim
             \dim_zero_new:N \l_@@_delta_y_two_dim}
```

The counter `\l_@@_instruction_int` will be the index of the loop over the instructions. The first value is 1.

```
       \int_zero_new:N \l_@@_instruction_int
       \int_incr:N \l_@@_instruction_int
```

We begin the loop over the instructions (the incrementation is at the end of the loop).

```
       \int_until_do:nNnn \l_@@_instruction_int > \g_@@_instruction_int
          {
```

We extract from the property list of the current instruction the fields "type", "row" and "column" and we store these values. We have to do a conversion because the components of a property list are token lists (and not integers).

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

```
          \prop_get:cnN {g_@@_instruction_\int_use:N \l_@@_instruction_int _prop}
                        {type} \l_tmpa_tl
          \int_set:Nn \l_@@_type_int {\l_tmpa_tl}
          \prop_get:cnN {g_@@_instruction_\int_use:N \l_@@_instruction_int _prop}
                        {row} \l_tmpa_tl
          \int_set:Nn \l_@@_row_int {\l_tmpa_tl}
          \prop_get:cnN {g_@@_instruction_\int_use:N \l_@@_instruction_int _prop}
                        {column} \l_tmpa_tl
          \int_set:Nn \l_@@_column_int {\l_tmpa_tl}
```

We fix the values of `\l_@@_di_int` and `\l_@@_dj_int` which indicate the direction of the dotted line to draw in the matrix.

```
           \int_case:nn \l_@@_type_int
             { 0 {\int_set:Nn \l_@@_di_int 0
                  \int_set:Nn \l_@@_dj_int 1}
               1 {\int_set:Nn \l_@@_di_int 0
                  \int_set:Nn \l_@@_dj_int 1}
               2 {\int_set:Nn \l_@@_di_int 1
                  \int_set:Nn \l_@@_dj_int 0}
               3 {\int_set:Nn \l_@@_di_int 1
                  \int_set:Nn \l_@@_dj_int 1}
               4 {\int_set:Nn \l_@@_di_int 1
                  \int_set:Nn \l_@@_dj_int {-1}}}
```

An instruction for a dotted line must have a initial cell and a final cell which are both not empty. If it's not the case, the instruction is said *impossible*. An error will be raised if an impossible instruction is encountered.

```
           \bool_if_exist:NTF \l_@@_impossible_instruction_bool
               {\bool_set_false:N \l_@@_impossible_instruction_bool}
               {\bool_new:N \l_@@_impossible_instruction_bool}
```

We will determine `\l_@@_final_i_int` and `\l_@@_final_j_int` which will be the "coordinates" of the end of the dotted line we have to draw.

```
           \int_zero_new:N  \l_@@_final_i_int
           \int_zero_new:N  \l_@@_final_j_int
           \int_set:Nn \l_@@_final_i_int \l_@@_row_int
           \int_set:Nn \l_@@_final_j_int \l_@@_column_int
           \bool_if_exist:NTF \l_@@_stop_loop_bool
                 {\bool_set_false:N \l_@@_stop_loop_bool}
                 {\bool_new:N \l_@@_stop_loop_bool}
           \bool_do_until:Nn \l_@@_stop_loop_bool
              {\int_add:Nn \l_@@_final_i_int \l_@@_di_int
               \int_add:Nn \l_@@_final_j_int \l_@@_dj_int
```

We test if we are still in the matrix.

```
               \bool_if:nTF { \int_compare_p:nNn \l_@@_final_i_int < \l_@@_number_of_first_row_int
                           || \int_compare_p:nNn \l_@@_final_i_int > \g_@@_row_int
                           || \int_compare_p:nNn \l_@@_final_j_int < 1
                           || \int_compare_p:nNn \l_@@_final_j_int > \g_@@_column_int}
```

If we are outside the matrix, the instruction is impossible and, of course, we stop the loop.

```
                       {\bool_set_true:N \l_@@_impossible_instruction_bool
                        \bool_set_true:N \l_@@_stop_loop_bool}
```

If we are in the matrix, we test if the cell is empty. If it's not the case, we stop the loop because we have found the correct values for `\l_@@_final_i_int` and `\l_@@_final_j_int`.

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

We will determine `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which will be the "coordinates" of the beginning of the dotted line we have to draw. The programmation is similar to the previous one.

```
            \int_zero_new:N  \l_@@_initial_i_int
            \int_zero_new:N  \l_@@_initial_j_int
            \int_set:Nn \l_@@_initial_i_int \l_@@_row_int
            \int_set:Nn \l_@@_initial_j_int \l_@@_column_int
```

If we know that the instruction is impossible (because it was not possible to found the correct value for `\l_@@_final_i_int` and `\l_@@_final_j_int`), we don't do this loop.

```
            \bool_set_eq:NN \l_@@_stop_loop_bool \l_@@_impossible_instruction_bool
            \bool_do_until:Nn \l_@@_stop_loop_bool
               {\int_sub:Nn \l_@@_initial_i_int \l_@@_di_int
                \int_sub:Nn \l_@@_initial_j_int \l_@@_dj_int
                \bool_if:nTF
                        {   \int_compare_p:nNn \l_@@_initial_i_int < \l_@@_number_of_first_row_int
                         || \int_compare_p:nNn \l_@@_initial_i_int > \g_@@_row_int
                         || \int_compare_p:nNn \l_@@_initial_j_int < 1
                         || \int_compare_p:nNn \l_@@_initial_j_int > \g_@@_column_int}
                        {\bool_set_true:N \l_@@_impossible_instruction_bool
                         \bool_set_true:N \l_@@_stop_loop_bool}
                        {\@@_if_not_empty_cell:nnT \l_@@_initial_i_int \l_@@_initial_j_int
                              {\bool_set_true:N \l_@@_stop_loop_bool}}
                }
```

Now, we can determine wether we have to draw a line. If the line is impossible, of course, we won't draw any line.

```
           \bool_if:NTF \l_@@_impossible_instruction_bool
             {\msg_error:nn {nicematrix} {Impossible~instruction}}
```

If the dotted line to draw is in the list of the previously drawn lines (`\l_@@_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.

```
             {\tl_set:Nx \l_tmpa_tl {\int_use:N \l_@@_initial_i_int-
                                     \int_use:N \l_@@_initial_j_int-
                                     \int_use:N \l_@@_final_i_int-
                                     \int_use:N \l_@@_final_j_int}
              \seq_if_in:NVF \l_@@_yet_drawn_seq \l_tmpa_tl
```

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

```
               {\seq_put_left:NV \l_@@_yet_drawn_seq \l_tmpa_tl
```

The four following variables are global because we will have to do affectations in a Tikz instruction (in order to extract the coordinates of two extremities of the line to draw).

```
               \dim_zero_new:N \g_@@_x_initial_dim
               \dim_zero_new:N \g_@@_y_initial_dim
               \dim_zero_new:N \g_@@_x_final_dim
               \dim_zero_new:N \g_@@_y_final_dim
```

We draw the line.

```
               \int_case:nn \l_@@_type_int
                {0  \@@_draw_ldots_line:
                 1  \@@_draw_cdots_line:
                 2  \@@_draw_vdots_line:
                 3  \@@_draw_ddots_line:
                 4  \@@_draw_iddots_line:}}}
```

Incrementation of the index of the loop (and end of the loop).

```
             \int_incr:N \l_@@_instruction_int
          }
      \group_begin:
      \tikzset{every~picture/.style = {overlay,
                                       remember~picture,
                                       name~prefix = nm-\int_use:N \g_@@_env_int-}}
```

```
    \cs_set_eq:NN \line \@@_line:nn
    \g_@@_code_after_tl
    \group_end:
}
```

The command `\@@_retrieve_coords:nn` retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw [14]. 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 anchors 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 `{pgfpicture}`.

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

```
\cs_new_protected:Nn \@@_draw_ldots_line:
     {\@@_retrieve_coords:nn {south~east} {south~west}
      \@@_draw_tikz_line:}
```

```
\cs_new_protected:Nn \@@_draw_cdots_line:
     {\@@_retrieve_coords:nn {mid~east} {mid~west}
      \@@_draw_tikz_line:}
```

For the vertical dotted lines, there is a problem because we want really vertical lines. If the type of the column is `c` (from a type `C` in `{NiceArray}`), all the Tikz nodes of the column have the same x-value for the anchors `south` and `north`. However, if the type of the column is `l` or `r` (from a type `L` or `R` in `{NiceArray}`), the geometric line from the anchors `south` and `north` would probably not be really vertical. That's why we need to known the type of the column and that's why we have constructed a token list `\g_@@_preamble_tl` to store the types (`l`, `c` or `r`) of all the columns.

```
\cs_new_protected:Nn \@@_draw_vdots_line:
     {\@@_retrieve_coords:nn {south} {north}
```

We store in `\t_tmpa_tl` the type of the column (`l`, `c` or `r`).

```
      \tl_set:Nx \l_tmpa_tl {\tl_item:Nn \g_@@_preamble_tl \l_@@_initial_j_int}
```

If the column is of type `l`, we will draw the dotted line on the left-most abscissa.

```
      \str_if_eq:VnT \l_tmpa_tl {l}
           {\dim_set:Nn \g_@@_x_initial_dim {\dim_min:nn \g_@@_x_initial_dim \g_@@_x_final_dim}
            \dim_set_eq:NN \g_@@_x_final_dim \g_@@_x_initial_dim}
```

If the column is of type `r`, we will draw the dotted line on the right-most abscissa.

```
      \str_if_eq:VnT \l_tmpa_tl {r}
           {\dim_set:Nn \g_@@_x_initial_dim {\dim_max:nn \g_@@_x_initial_dim \g_@@_x_final_dim}
            \dim_set_eq:NN \g_@@_x_final_dim \g_@@_x_initial_dim}
```

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

Now, the coordinates of the line to draw are computed in \g_@@_x_initial_dim, \g_@@_y_initial_dim, \g_@@_x_final_dim and \g_@@_y_final_dim. We can draw the line with \l_@@_draw_tikz_line: as usual.

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

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

```
\cs_new_protected:Nn \@@_draw_ddots_line:
   {\@@_retrieve_coords:nn {south~east} {north~west}
```

We have retrieved the coordinates in the usual way (they are stored in \g_@@_x_initial_dim, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
       {\int_incr:N \l_@@_ddots_int
```

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

```
        \int_compare:nNnTF \l_@@_ddots_int = 1
```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```
          {\dim_set:Nn \l_@@_delta_x_one_dim {\g_@@_x_final_dim - \g_@@_x_initial_dim }
           \dim_set:Nn \l_@@_delta_y_one_dim {\g_@@_y_final_dim - \g_@@_y_initial_dim }}
```

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate \g_@@_y_initial_dim.

```
          {\dim_gset:Nn \g_@@_y_final_dim
                  {\g_@@_y_initial_dim +
                      (\g_@@_x_final_dim - \g_@@_x_initial_dim)
                      * \dim_ratio:nn \l_@@_delta_y_one_dim \l_@@_delta_x_one_dim }}}
```

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

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

We draw the \Iddots diagonals in the same way.

```
\cs_new_protected:Nn \@@_draw_iddots_line:
   {\@@_retrieve_coords:nn {south~west} {north~east}
    \bool_if:NT \l_@@_parallelize_diags_bool
       {\int_incr:N \l_@@_iddots_int
        \int_compare:nNnTF \l_@@_iddots_int = 1
          {\dim_set:Nn \l_@@_delta_x_two_dim {\g_@@_x_final_dim - \g_@@_x_initial_dim }
           \dim_set:Nn \l_@@_delta_y_two_dim {\g_@@_y_final_dim - \g_@@_y_initial_dim }}
          {\dim_gset:Nn \g_@@_y_final_dim
                  {\g_@@_y_initial_dim +
                      (\g_@@_x_final_dim - \g_@@_x_initial_dim)
                      * \dim_ratio:nn \l_@@_delta_y_two_dim \l_@@_delta_x_two_dim }}}
    \@@_draw_tikz_line:}
```

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.

```
\cs_new_protected:Nn \@@_draw_tikz_line:
                {
```

The dimension \l_@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of expl3 to compute this length.

```
                 \dim_zero_new:N \l_@@_l_dim
                 \dim_set:Nn \l_@@_l_dim
                            { \fp_to_dim:n
                                { sqrt( (  \dim_use:N \g_@@_x_final_dim
                                          -\dim_use:N \g_@@_x_initial_dim) ^2
```

```
                                     +(  \dim_use:N \g_@@_y_final_dim
                                        -\dim_use:N \g_@@_y_initial_dim) ^2 )}
                               }
```

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

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

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

```
                    {\int_set:Nn \l_tmpa_int {\dim_ratio:nn {\l_@@_l_dim - 0.54em}
                                                            {0.45em}}
```

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

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

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.

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

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).

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

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

```
\cs_new_protected:Nn \@@_add_to_empty_cells:
    {\seq_gput_right:Nx \g_@@_empty_cells_seq
          {\int_use:N \g_@@_row_int-
           \int_use:N \g_@@_column_int}}
```

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of nicematrix rely upon `{NiceArray}`).

```
\NewDocumentCommand \@@_Ldots {s}
    {\IfBooleanF {#1} {\@@_instruction_of_type:n 0}
     \bool_if:NF \l_@@_nullify_dots_bool {\phantom \@@_ldots}
     \@@_add_to_empty_cells:}
```

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

```
\NewDocumentCommand \@@_Vdots {s}
    {\IfBooleanF {#1} {\@@_instruction_of_type:n 2}
     \bool_if:NF \l_@@_nullify_dots_bool {\phantom \@@_vdots}
     \@@_add_to_empty_cells:}
```

```
\NewDocumentCommand \@@_Ddots {s}
    {\IfBooleanF {#1} {\@@_instruction_of_type:n 3}
     \bool_if:NF \l_@@_nullify_dots_bool {\phantom \@@_ddots}
     \@@_add_to_empty_cells:}
```

```
\NewDocumentCommand \@@_Iddots {s}
    {\IfBooleanF {#1} {\@@_instruction_of_type:n 4}
     \bool_if:NF \l_@@_nullify_dots_bool {\phantom \@@_iddots}
     \@@_add_to_empty_cells:}
```

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

```
\cs_new_protected:Nn \@@_Hspace:
  {\@@_add_to_empty_cells:
   \hspace}
```

The command `\@@_NiceMatrixEndPoint:` will be linked to `\NiceMatrixEndPoint` in `{NiceArray}`.

```
\cs_new_protected:Nn \@@_NiceMatrixEndPoint:
     {\kern 0.5pt}
```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArray}`. This command uses an optionnal argument like `\hdotsfor` but this argument is discarded (in `\hdotsfor`, this argument is used for fine tuning of the space beetween two consecutive dots).

```
\NewDocumentCommand \@@_Hdotsfor {O{} m}
```

First, we store in the `code-after` (that is in `\g_@@_code_after_tl`) a command `\line` to draw the dotted line after the construction of the matrix.

```
               {\int_set:Nn \l_tmpa_int {\g_@@_column_int - 1}
                \int_set:Nn \l_tmpb_int {\g_@@_column_int + #2}
                \tl_gput_right:Nx \g_@@_code_after_tl
                       {\exp_not:N \line {\int_use:N\g_@@_row_int-\int_use:N\l_tmpa_int}
                                         {\int_use:N\g_@@_row_int-\int_use:N\l_tmpb_int}}
```

Then, we insert the correct number of ampersands (`&`).

```
                \prg_replicate:nn {#2-1} {&}}
```

Tikz nodes are created for all the cells of the array, even the implicit cells of the `\Hdotsfor`.

12.8 We process the options

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.
We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` execute the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.
Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

```
\ProcessKeysOptions {NiceMatrix}
```

12.9 The command \line accessible in code-after

In the `code-after`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specification of two cells in the array (in the format *i*-*j*) and draws a dotted line between these cells.

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

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

12.10 The error messages

```
\msg_new:nnnn {nicematrix}
              {Impossible~instruction}
              {It's~not~possible~to~execute~the~instruction~
               \int_case:nn \l_@@_type_int
                 {0 {\token_to_str:N \Ldots}
                  1 {\token_to_str:N \Cdots}
                  2 {\token_to_str:N \Vdots}
                  3 {\token_to_str:N \Ddots}}~in~the~row~\int_use:N\l_@@_row_int\
               ~and~the~column~\int_use:N\l_@@_column_int\space of~the~matrix~
               because~it's~impossible~to~find~one~of~its~extremities~
               (both~extremities~must~be~non~empty~cells~of~the~matrix).~
               If~you~go~on,~the~instruction~will~be~ignored.}
              {You~can~specify~an~end~of~line~on~a~empty~cell~
               with~\token_to_str:N \NiceMatrixEndPoint.}
\msg_new:nnn {nicematrix}
             {Multicolumn~forbidden}
             {The~command~\token_to_str:N \multicolumn\
              is~forbidden~in~the~environment~\{\@currenvir\}~
              and~its~variants.~The~command~\token_to_str:N \hdotsfor\
              of~amsmath~is~also~forbidden~since~it~uses~
              \token_to_str:N \multicolumn\ ~(but~you~can~use~\token_to_str:N \Hdotsfor\
              ~instead).~You~can~go~on~but~your~matrix~may~be~wrong.}
\msg_new:nnn {nicematrix}
             {We~are~yet~in~an~environment~NiceArray}
             {Environments~\{NiceArray\}~(or~\{NiceMatrix\},~etc.)~can't~be~
              nested.~We~can~go~on,~but,~maybe,~you~will~have~errors~or~an~incorrect~
              result.}
\msg_new:nnnn {nicematrix}
              {Duplicate~name}
              {The~name~"#1"~is~already~used~and~you~shouldn't~use~
               the~same~environment~name~twice.~You~can~go~on,~but,~
               maybe,~you~will~have~incorrect~results~especially~
               if~you~use~"columns-width=auto".\\
               For~a~list~of~the~names~already~used,~type~H~<return>.}
              {The~names~already~defined~in~this~document~are:~
               \seq_use:Nnnn~\g_@@_names_seq~{,~} {,~} {~and~}.}
```

12.11 The environment {NiceMatrixBlock}

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

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

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

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

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

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

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

12.12 The environment {pNiceArrayC} and its variants

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

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

```
\keys_define:nn {NiceMatrix/NiceArrayC}
     {parallelize-diags .bool_set:N       = \l_@@_parallelize_diags_bool,
      parallelize-diags .default:n        = true,
      renew-dots        .bool_set:N       = \l_@@_renew_dots_bool,
      renew-dots        .default:n        = true,
      nullify-dots      .bool_set:N       = \l_@@_nullify_dots_bool ,
      nullify-dots      .default:n        = true,
      code-for-last-col .tl_set:N         = \l_@@_code_for_last_col_tl,
      code-for-last-col .value_required:n = true,
      columns-width     .code:n           = \str_if_eq:nnTF {#1} {auto}
                                              {\bool_set_true:N
                                                     \l_@@_auto_columns_width_bool}
                                              {\dim_set:Nn \l_@@_columns_width_dim {#1}},
      columns-width     .value_required:n = true,
      name              .code:n           = {\seq_if_in:NnTF \g_@@_names_seq {#1}
                                                {\msg_error:nnn {nicematrix}
                                                               {Duplicate~name}
                                                               {#1}}
                                                {\seq_gput_left:Nn \g_@@_names_seq {#1}}
                                              \tl_gset:Nn \g_@@_name_tl {#1}},
      name              .value_required:n = true,
      code-after        .tl_gset:N         = \g_@@_code_after_tl,
      code-after        .initial:n    = \c_empty_tl,
      code-after        .value_required:n = true,
      unknown .code:n  = \msg_error:nn {nicematrix} {Unknown~option~for~NiceArrayC}}
```

```
\msg_new:nnnn {nicematrix}
              {Unknown~option~for~NiceArrayC}
              {The~option~"\tl_use:N\l_keys_key_tl"~is~unknown~for~the~environment~
               \{\@currenvir\}.\\
               If~you~go~on,~it~will~be~ignored.\\
               For~a~list~of~the~available~options,~type~H~<return>.}
              {The~available~options~are~(in~alphabetic~order):~
               code-after,~
               code-for-last-col,~
               columns-width,~
               name,~
               nullify-dots,~
               parallelize-diags~
               and~renew-dots.}
```

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

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

```
\cs_new_protected:Nn \@@_NiceArrayC:n
    {
     \begin{NiceArray}
```

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

```
             {#1
```

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

```
                >{\int_gincr:N \g_@@_column_int
                  \hbox_set:Nw \l_tmpa_box
                  \tl_if_empty:NTF \g_@@_name_tl
                   {\tikz[remember~picture, inner~sep = 0pt, minimum~width = 0pt, baseline]
                       \node [anchor=base,
                              name = nm-\int_use:N \g_@@_env_int-
                                        \int_use:N \g_@@_row_int-
                                        \int_use:N \g_@@_column_int] }
                   {\tikz[remember~picture, inner~sep = 0pt, minimum~width = 0pt, baseline]
                       \node [anchor=base,
                              name = nm-\int_use:N \g_@@_env_int-
                                        \int_use:N \g_@@_row_int-
                                        \int_use:N \g_@@_column_int,
                              alias = \g_@@_name_tl-
                                      \int_use:N \g_@@_row_int-
                                      \int_use:N \g_@@_column_int] }
                     \bgroup $ % $
                     \l_@@_code_for_last_col_tl
                 }
                l
```

We actualize the value of \g_@@_with_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).

```
                <{   $ % $
                     \egroup ;
                  \hbox_set_end:
                  \dim_gset:Nn \g_@@_width_last_col_dim
                     {\dim_max:nn \g_@@_width_last_col_dim
                                  {\box_wd:N \l_tmpa_box}}
                  \tl_gput_right:Nn \g_@@_preamble_aux_tl {#1}
                  \skip_horizontal:n {-2\arraycolsep}
```

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

```
                  \hbox_overlap_right:n
                       {\skip_horizontal:n {2\arraycolsep}
                        \box_use:N \l_tmpa_box}}}}
```

This ends the preamble of the array that will be constructed (a rather long preamble, indeed).

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

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

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

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

12.13 The environment {pNiceArrayRC}

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

If the environment is used with the option `columns-width=auto`, the construction

```
\begin{pNiceArrayRC}{preamble}[columns-width=auto]
 first row \\
 second row \\
 third row \\
 etc. \\
\end{pNiceArrayRC}
```

is transformed in the following construction :

```
\begin{NiceMatrixBlock}[auto-columns-width]
\vbox{\hbox{\hphantom{an open parenthesis}
            $\begin{NiceArray}{preamble}
              first row \\
             \end{NiceArray}$}
      \hbox{$\begin{pNiceArrayC}{preamble}
              second row \\
              third row \\
              etc. \\
             \end{pNiceArrayC}$}}
\end{NiceMatrixBlock}
```

If the environment is used with a numerical value for the option `columns-width`, the construction

```
\begin{pNiceArrayRC}{preamble}[columns-width=value]
 first row \\
 second row \\
 third row \\
 etc. \\
\end{pNiceArrayRC}
```

is transformed in the following contruction :

```
\NiceMatrixOptions{columns-width=value}
\vbox{\hbox{skip of the width of a parenthesis
            $\begin{NiceArray}{preamble}
              first row \\
             \end{NiceArray}$}
      \hbox{$\begin{pNiceArrayC}{preamble}
              second row \\
              third row \\
              etc. \\
             \end{pNiceArrayC}$}}
```

(in the group created by the environment).

```
\keys_define:nn {NiceMatrix/NiceArrayRC}
 {parallelize-diags    .bool_set:N = \l_@@_parallelize_diags_bool,
  parallelize-diags    .default:n  = true,
  renew-dots           .bool_set:N = \l_@@_renew_dots_bool,
  renew-dots           .default:n  = true,
  nullify-dots         .bool_set:N = \l_@@_nullify_dots_bool ,
  nullify-dots         .default:n  = true,
  code-for-first-row   .tl_set:N   = \l_@@_code_for_first_row_tl,
  code-for-last-col    .tl_set:N   = \l_@@_code_for_last_col_tl,
  code-for-last-col    .value_required:n = true,
  columns-width        .tl_set:N   = \l_@@_columns_width_tl,
  name                 .code:n     = {\seq_if_in:NnTF \g_@@_names_seq {#1}
                                        {\msg_error:nnn {nicematrix}
                                                        {Duplicate~name}
                                                        {#1}}
                                        {\seq_gput_left:Nn \g_@@_names_seq {#1}}
                                      \tl_gset:Nn \l_@@_name_tl {#1}},
  code-after           .tl_set:N   = \l_@@_code_after_tl,
  unknown .code:n  = \msg_error:nn {nicematrix} {Unknown~option~for~NiceArrayRC}}
```

```
\msg_new:nnnn {nicematrix}
              {Unknown~option~for~NiceArrayRC}
              {The~option~"\tl_use:N\l_keys_key_tl"~is~unknown~for~the~environment~
               \{\@currenvir\}.\\
               If~you~go~on,~it~will~be~ignored.\\
               For~a~list~of~the~available~options,~type~H~<return>.}
              {The~available~options~are~(in~alphabetic~order):~
               code-after,~
               code-for-last-col,~
               code-for-first-row,~
               columns-width~(in~fact~mandatory),~
               name,~
               nullify-dots,~
               parallelize-diags~
               and~renew-dots.}
```

```
\msg_new:nnn {nicematrix}
               {The~key~columns-width~is~mandatory}
               {In~the~environment~\{\@currenvir\}~you~must~use~the~
                key~"columns-width"~either~with~a~numerical~value~
                or~the~special~value~"auto".\\
                If~you~go~on,~the~value~"auto"~will~be~used~for~
                "columns-width".}
```

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

```
\NewDocumentEnvironment{NiceArrayRCwithDelims} {mm O{} m O{}}
    {\tl_clear_new:N \l_@@_columns_width_tl
     \tl_clear_new:N \l_@@_name_tl
     \tl_clear_new:N \l_@@_code_after_tl
     \keys_set:nn {NiceMatrix/NiceArrayRC} {#3,#5}
     \tl_gset_eq:NN \g_@@_name_tl \l_@@_name_tl
     \tl_if_empty:NT \l_@@_columns_width_tl
             {\msg_error:nn {nicematrix} {The~key~columns-width~is~mandatory}
              \tl_set:Nn \l_@@_columns_width_tl {auto}}
     \str_if_eq:nVTF {auto} \l_@@_columns_width_tl
        {\begin{NiceMatrixBlock}[auto-columns-width]}
        {\NiceMatrixOptions{columns-width = \l_@@_columns_width_tl} }
     \vbox\bgroup\hbox\bgroup
```

Here, it would be possible to use a \hphantom but we prefer to use expl3 instructions.

```
     \hbox_set:Nn \l_tmpa_box {$\left#1\vbox_to_ht:nn{2cm}{}\right.$}
     \skip_horizontal:n {\box_wd:N \l_tmpa_box - \nulldelimiterspace}
```

We give the value -1 to \l_@@_number_of_first_row_int and, therefore, the first row of the array will have the number 0 (the final user will use this value when using, for example, the command \line in code-after).

```
     \int_set:Nn \l_@@_number_of_first_row_int {-1}
     \tl_set_eq:NN \l_@@_code_for_cell_tl \l_@@_code_for_first_row_tl
```

We read staticly the first row of the array given by the user because we will use the first row to construct an environment {NiceArray}. *We read until the first control sequence* \\. Therefore, it's not possible to use a sub-array with this control sequence in the first row of the array.

```
     \cs_set:Npn \@@_replace_first_line ##1\\{
           $\begin{NiceArray}{*\c@MaxMatrixCols{C}}
             ##1
            \end{NiceArray}$
           \egroup \hbox\bgroup
```

We decrement the number of the environment because we want the same number for both constructed environments.

```
           \int_gdecr:N \g_@@_env_int
           \tl_gset_eq:NN \g_@@_name_tl \l_@@_name_tl
           \tl_gset_eq:NN \g_@@_code_after_tl \l_@@_code_after_tl
           \bool_set_false:N \l_@@_direct_NiceArray_bool
           \dim_gzero_new:N \g_@@_width_last_col_dim
           \bool_set_false:N \l_@@_exterior_arraycolsep_bool
           \tl_set:Nn \l_@@_pos_env_tl c
```

```
          $\left#1 % $
          \@@_NiceArrayC:n {#4}}
    \@@_replace_first_line
   }
   {\end{NiceArray}
    \right#2$ % $
    \skip_horizontal:n \g_@@_width_last_col_dim
    \egroup\egroup
    \str_if_eq:nVT {auto} \l_@@_columns_width_tl
       {\end{NiceMatrixBlock}} }
```

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

```
\NewDocumentEnvironment {pNiceArrayRC} {}
     {\NiceArrayRCwithDelims{(}{)}}
     {\endNiceArrayRCwithDelims}
\NewDocumentEnvironment {bNiceArrayRC} {}
     {\NiceArrayRCwithDelims{[}{]}}
     {\endNiceArrayRCwithDelims}
\NewDocumentEnvironment {vNiceArrayRC} {}
     {\NiceArrayRCwithDelims{|}{|}}
     {\endNiceArrayRCwithDelims}
\NewDocumentEnvironment {VNiceArrayRC} {}
     {\NiceArrayRCwithDelims{\|}{\|}}
     {\endNiceArrayRCwithDelims}
\NewDocumentEnvironment {BNiceArrayRC} {}
     {\NiceArrayRCwithDelims{\{}{\}}}
     {\endNiceArrayRCwithDelims}
```

13 History

13.1 Changes between versions 1.0 and 1.1

Option `silent` (with this option, the impossible instructions are discarded silently).
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.
We decrement `\g_@@_row_int` in `\@@_draw_lines:` to have the exact number of rows.
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.