The package \texttt{nicematrix}*

F. Pantigny  
fpantigny@wanadoo.fr  
April 6, 2020

Abstract

The LaTeX package \texttt{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 \texttt{xelatex}, \texttt{lualatex}, \texttt{pdflatex} but also by the classical workflow \texttt{latex-dvips-ps2pdf} (or Adobe Distiller). Two or three compilations may be necessary. This package requires and loads the packages \texttt{l3keys2e}, \texttt{xparse}, \texttt{array}, \texttt{amsmath}, \texttt{pgfcore} and the module \texttt{shapes} of PGF (\texttt{tikz} is not loaded). The final user only has to load the package with \texttt{\usepackage[nicematrix]}.

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

- continuous dotted lines\textsuperscript{1};
- exterior rows and columns for labels;
- a control of the width of the columns.

A command \texttt{\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 \texttt{\pmatrix} of \texttt{amsmath}.

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

This code composes the matrix $A$ on the right.

Now, if we use the package \texttt{nicematrix} with the option \texttt{transparent}, the same code will give the result on the right.

\texttt{\NiceMatrixOptions{transparent}}

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

\texttt{\NiceMatrixOptions{}}

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

\texttt{\NiceMatrixOptions{}}

\textsuperscript{1}This document corresponds to the version 3.15 of \texttt{nicematrix}, at the date of 2020/04/06.

\textsuperscript{1}If the class option \texttt{draft} is used, these dotted lines will not be drawn for a faster compilation.
2 The environments of this package

The package \texttt{nicematrix} defines the following new environments.

\begin{quote}
\{NiceMatrix\} \quad \{NiceArray\} \quad \{pNiceArray\}
\{bNiceMatrix\} \quad \{vNiceMatrix\} \quad \{BNiceMatrix\} \quad \{vNiceMatrix\}
\{pNiceMatrix\} \quad \{bNiceArray\} \quad \{BNiceArray\} \quad \{BNiceMatrix\}
\end{quote}

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

The environment \{NiceArray\} is similar to the environment \{array\} of the package \texttt{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. 8 the section relating to \{NiceArray\}.

3 The continuous dotted lines

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

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 on both sides of the current cell. Of course, for \texttt{\Ldots} and \texttt{\Cdots}, it's an horizontal line; for \texttt{\Vdots}, it's a vertical line and for \texttt{\Ddots} and \texttt{\Iddots} diagonal ones. It’s possible to change the color of these lines with the option \texttt{color}.

\begin{verbatim}
\begin{bNiceMatrix}
 a_1 & \Cdots & \ & \ & a_1 \\
 \Vdots & a_2 & \Cdots & \ & a_2 \\
 & \ & \ & \ & \Vdots[\text{color=red}] \\
 a_1 & a_2 & \ & \ & a_n
\end{bNiceMatrix}
\end{verbatim}

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

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

\footnote{However, for the columns of type w and W, the cells are composed in math mode (in the environments of \texttt{nicematrix}) whereas in \texttt{array} of \texttt{array}, they are composed in text mode.}

\footnote{The command \texttt{\iddots}, defined in \texttt{nicematrix}, is a variant of \texttt{\dots} with dots going forward. If \texttt{mathdots} is loaded, the version of \texttt{mathdots} is used. It corresponds to the command \texttt{\adots} of \texttt{unicode-math}.}

\footnote{The precise definition of a “non-empty cell” is given below (cf. p. 19).}

\footnote{It’s also possible to change the color of all theses dotted lines with the option \texttt{xdots/color} (\texttt{xdots} to remind that it works for \texttt{\Cdots}, \texttt{\Ldots}, \texttt{\Vdots}, etc.): cf. p. 5.}
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 \texttt{nicematrix}:

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

\[
\begin{bNiceMatrix}
0 & \Cdots & \Cdots & 0 \\
\Vdots & & & \Vdots \\
\Vdots & & & \Vdots \\
0 & \Cdots & \Cdots & 0
\end{bNiceMatrix}
\]

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

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

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

\[
\begin{bNiceMatrix}
0 & \Cdots & & 0 \\
\Vdots & & & \Vdots \\
& & & \Vdots \\
0 & & \Cdots & 0
\end{bNiceMatrix}
\]

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command \texttt{\hspace} for the vertical dimension and a command \texttt{\hspace*} in a cell for the horizontal dimension.\footnote{However, a command \texttt{\hspace*} might interfere with the construction of the dotted lines. That’s why the package \texttt{nicematrix} provides a command \texttt{\Hspace} which is a variant of \texttt{\hspace} transparent for the dotted lines of \texttt{nicematrix}.}

However, a command \texttt{\hspace*} might interfere with the construction of the dotted lines. That’s why the package \texttt{nicematrix} provides a command \texttt{\Hspace} which is a variant of \texttt{\hspace} transparent for the dotted lines of \texttt{nicematrix}.

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

\[
\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 classically with the environment \texttt{pmatrix} of \texttt{amsmath}.

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

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

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

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

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

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

\[
C = \begin{pmatrix}
h & i & j & k & l & m \\
x & \ldots & \ldots & \ldots & \ldots & x
\end{pmatrix}
\]

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

\footnote{And it’s not possible to draw a \texttt{\Ldots} and a \texttt{\Cdots} line between the same cells.}

\footnote{In \texttt{nicematrix}, one should use \texttt{\Hspace*} and not \texttt{\hspace} for such an usage because \texttt{nicematrix} loads \texttt{array}. One may also remark that it’s possible to fix the width of a column by using the environment \texttt{NiceArray} (or one of its variants) with a column of type \texttt{w} or \texttt{W}; see p. 12}
However, one may prefer the geometry of the first matrix $A$ and would like to have such a geometry with a dotted line in the second row. It’s possible by using the option `nullify-dots` (and only one instruction `{\Ldots}` is necessary).

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

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

There must be no space before the opening bracket `{` of the options of the environment.

### 3.2 The command `{\Hdotsfor}`

Some people commonly use the command `{\Hdotsfor}` of `amsmath` in order to draw horizontal dotted lines in a matrix. In the environments of `nicematrix`, one should use instead `{\Hdotsfor}` in order to draw dotted lines similar to the other dotted lines drawn by the package `nicematrix`. As with the other commands of `nicematrix` (like `{\Cdots}`, `{\Ldots}`, `{\Vdots}`, etc.), the dotted line drawn with `{\Hdotsfor}` extends until the contents of the cells on both sides.

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

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

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

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

### 3.3 How to generate the continuous dotted lines transparently

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

- **The option `renew-dots`**
  
  With this option, the commands `{\ldots}`, `{\cdots}`, `{\vdots}`, `{\ddots}`, `{\iddots}` and `{\Hdotsfor}` are redefined within the environments provided by `nicematrix` and behave like `{\Ldots}`, `{\Cdots}`, `{\Vdots}`, `{\Ddots}` and `{\Hdotsfor}`; the command `{\dots}` (“automatic dots” of `amsmath`) is also redefined to behave like `{\Ldots}`.

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

---

8 The options `renew-dots`, `renew-matrix` and `transparent` can be fixed with the command `{\NiceMatrixOptions}` like the other options. However, they can also be fixed as options of the command `{\usepackage}` (it’s an exception for these three specific options.)
Therefore, with the option transparent, a classical code gives directly the output of nicematrix.

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

3.4 The labels of the dotted lines

The commands \Ldots, \Cdots, \Vdots, \Ddots, \Iddots and \Hdotsfor (and the command \line in the code-after which is described p. 7) accept two optional arguments specified by the tokens _ and ^ for labels positioned below and above the line. The arguments are composed in math mode with \scriptstyle.

\begin{bNiceMatrix}
1 & \hspace*{1cm} & 0 \\
& & \\
0 & & 1
\end{bNiceMatrix}

3.5 Customization of the dotted lines

The dotted lines drawn by \Ldots, \Cdots, \Vdots, \Ddots, \Iddots and \Hdotsfor (and by the command \line in the code-after which is described p. 7) may be customized by three options (specified between square brackets after the command):

* color;
* shorten;
* line-style.

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

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

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

The option xdots/color

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

The option xdots/shorten

The option xdots/shorten fixes the margin of both extremities of the line. The name is derived from the options “shorten >” and “shorten <” of Tikz but one should notice that nicematrix only provides xdots/shorten. The initial value of this parameter is 0.3 cm (it is recommended to use a unit of length dependent of the current font).

The option xdots/line-style

It should be pointed that, by default, the lines drawn by Tikz with the parameter dotted are composed of square dots (and not rounded ones).\footnote{The first reason of this behaviour is that the PDF format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the PDF readers. It’s easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded dots needs a specification of each dot in the PDF file.}
In order to provide lines with rounded dots in the style of those provided by \ldots (at least with the Computer Modern fonts), the package \nicematrix embeds its own system to draw a dotted line (and this system uses PGF and not Tikz). This style is called \texttt{standard} and that’s the initial value of the parameter \texttt{xdots/line-style}.

However (when Tikz is loaded) it’s possible to use for \texttt{xdots/line-style} any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tikz pathes (with the exception of “color”, “shorten >” and “shorten <”).

Here is for example a tridiagonal matrix with the style \texttt{loosely dotted}:

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

\[\begin{pmatrix}
  a & b & 0 & \ldots & 0 \\
  b & a & b & \ldots & \\
  0 & b & a & \ldots & 0 \\
  \ldots & \ldots & 0 & b \\
  0 & \ldots & 0 & b & a
\end{pmatrix}\]

\section{The PGF/Tikz nodes created by \nicematrix}

The package \nicematrix creates a PGF/Tikz node for each (non-empty) cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix. However, the user may wish to use directly these nodes. It’s possible (if Tikz has been loaded\(^\dagger\)). First, the user have to give a name to the array (with the key called \texttt{name}). Then, the nodes are accessible through the names \texttt{name-\texttt{i-j}} where \texttt{name} is the name given to the array and \texttt{i} and \texttt{j} the numbers of the row and the column of the considered cell.

\[
\begin{pNiceMatrix}[name=mymatrix]
\begin{pmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9
\end{pmatrix}
\end{pNiceMatrix}
\]

\begin{tikzpicture}[remember picture,overlay]
  \draw (mymatrix-2-2) circle (2mm);
\end{tikzpicture}

Don’t forget the options \texttt{remember picture} and \texttt{overlay}.

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

\[
\begin{pmatrix}
  a & a+b & a+b+c \\
  a & a & a+b \\
  a & a & a
\end{pmatrix}
\]

In fact, the package \nicematrix can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option \texttt{create-medium-nodes} and the second ones with the option \texttt{create-large-nodes}.\(^\dagger\)

\(^\dagger\)We remind that, since the version 3.13, \nicematrix doesn’t load Tikz by default by only PGF (Tikz is a layer over PGF).

\(^\dagger\)There is also an option \texttt{create-extra-nodes} which is an alias for the conjunction of \texttt{create-medium-nodes} and \texttt{create-large-nodes}.
The names of the “medium nodes” are constructed by adding the suffix “-medium” to the names of the “normal nodes”. In the following example, we have underlined the “medium nodes”. We consider that this example is self-explanatory.

\[
\begin{pmatrix}
a & a + b & a + b + c \\
a & a & a + b \\
a & a & a \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
a & a + b & a + b + c \\
a & a & a + b \\
a & a & a \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
a & a + b & a + b + c \\
a & a & a + b \\
a & a & a \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
a & a + b & a + b + c \\
a & a & a + b \\
a & a & a \\
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
a & a + b & a + b + c \\
a & a & a + b \\
a & a & a \\
\end{pmatrix}
\]

We explain below how to fill the nodes created by \texttt{nicematrix} (cf. p. 23).

### 5 The code-after

The option \texttt{code-after} may be used to give some code that will be executed after the construction of the matrix (and thus after the construction of all the nodes).

If Tikz is loaded\textsuperscript{14}, one may access to that nodes with classical Tikz instructions. The nodes should be designed as $i$-$j$ (without the prefix corresponding to the name of the environment).

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

\textsuperscript{12}There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 10).

\textsuperscript{13}The options \texttt{left-margin} and \texttt{right-margin} take dimensions as values but, if no value is given, the default value is used, which is \texttt{\arraycolsep} (by default: 5 pt). There is also an option \texttt{margin} to fix both \texttt{left-margin} and \texttt{right-margin} to the same value.

\textsuperscript{14}We remind that, since the version 3.13, \texttt{nicematrix} doesn’t load Tikz by default but only \texttt{pgf} (Tikz is a layer over \texttt{pfg}).
The environment \{NiceArray\} and its variants

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 \texttt{L}, \texttt{C} and \texttt{R} instead of \texttt{l}, \texttt{c} and \texttt{r}. It’s possible to use the constructions \texttt{w(...){...}, W(...){...}, I, >{...}, <{...}, \{...\}, !{...} and *{n}{...} but the letters \texttt{p}, \texttt{m} and \texttt{b} should not be used.

The package \texttt{nicematrix} provides also the variants \{pNiceArray\}, \{vNiceArray\}, \{VNiceArray\}, \{bNiceArray\} and \{BNiceArray\}.

Of course, one of the benefits of \{pNiceArray\} over \{pNiceMatrix\} is the possibility of drawing vertical rules:

\[
\left[\begin{NiceArray}{CCCC|C}
\arraycolsep=1.4pt
a_1 & ? & \Cdots & ? & ? \\
0 & & \Ddots & \Vdots & \Vdots \\
\Vdots & \Ddots & \Ddots & ? \\
0 & \Cdots & 0 & a_n & ? \\
\end{NiceArray}\right]
\]

Another benefit is the possibility of using different alignments of columns.

\[
\begin{pNiceArray}{LCR}
a_{(1)} & \Cdots & a_{(1n)} \\
a_{(2)} & & a_{(2n)} \\
\Vdots & & \Vdots \\
a_{(n-1,1)} & \Cdots & a_{(n-1,n)} \\
\end{pNiceArray}
\]

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

\begin{NiceMatrix}[code-after=\line{2-2}{3-3}]
I & 0 & \Cdots & 0 \\
0 & I & \Ddots & \Vdots \\
\Vdots & \Ddots & I & 0 \\
0 & \Cdots & 0 & I \\
\end{NiceMatrix}

For the readability of the code, an alternative syntax is provided: it’s possible to give the instructions of the \texttt{code-after} at the end of the environment, after the keyword \texttt{CodeAfter}.

\begin{NiceMatrix}[code-after=\line{2-2}{3-3}]
I & 0 & \Cdots & 0 \\
0 & I & \Ddots & \Vdots \\
\Vdots & \Ddots & I & 0 \\
0 & \Cdots & 0 & I \\
\end{NiceMatrix}

\section{The environment \{NiceArray\} and its variants}

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 \texttt{L}, \texttt{C} and \texttt{R} instead of \texttt{l}, \texttt{c} and \texttt{r}. It’s possible to use the constructions \texttt{w(...){...}, W(...){...}, I, >{...}, <{...}, \{...\}, !{...} and *{n}{...} but the letters \texttt{p}, \texttt{m} and \texttt{b} should not be used.

The package \texttt{nicematrix} provides also the variants \{pNiceArray\}, \{vNiceArray\}, \{VNiceArray\}, \{bNiceArray\} and \{BNiceArray\}.

Of course, one of the benefits of \{pNiceArray\} over \{pNiceMatrix\} is the possibility of drawing vertical rules:

\[
\left[\begin{NiceArray}{CCCC|C}
\arraycolsep=1.4pt
a_1 & ? & \Cdots & ? & ? \\
0 & & \Ddots & \Vdots & \Vdots \\
\Vdots & \Ddots & \Ddots & ? \\
0 & \Cdots & 0 & a_n & ? \\
\end{NiceArray}\right]
\]

Another benefit is the possibility of using different alignments of columns.

\[
\begin{pNiceArray}{LCR}
a_{(1)} & \Cdots & a_{(1n)} \\
a_{(2)} & & a_{(2n)} \\
\Vdots & & \Vdots \\
a_{(n-1,1)} & \Cdots & a_{(n-1,n)} \\
\end{pNiceArray}
\]

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

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

In a command \texttt{\multicolumn}, one should also use the letters \texttt{L}, \texttt{C}, \texttt{R}.

\begin{NiceMatrix}[code-after=\line{2-2}{3-3}]
I & 0 & \Cdots & 0 \\
0 & I & \Ddots & \Vdots \\
\Vdots & \Ddots & I & 0 \\
0 & \Cdots & 0 & I \\
\end{NiceMatrix}
$\begin{NiceArrayWithDelims}
\{\downarrow\}\{\uparrow\}\{\text{CCC}\}[\text{margin}]
1 & 2 & 3 \ \ \ \\
4 & 5 & 6 \ \ \\
7 & 8 & 9 \ \ 
\end{NiceArrayWithDelims}$

7 The vertical position of the arrays

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

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

It’s also possible to use the option baseline with one of the special values t, c or b. These letters may also be used absolutely like the option of the environment {array} of array. The initial value of baseline is c.

In the following example, we use the option t (equivalent to baseline=t) immediately after an item of list. One should remark that the presence of a \hline at the beginning of the array doesn’t prevent the alignment of the baseline with the baseline of the first row (with {array} of array, one must use \firsthline\).

\begin{enumerate}
\item an item
\item \renewcommand{\arraystretch}{1.2}
$\begin{NiceArray}[t]{LCCCCCC}
\hline
n & 0 & 1 & 2 & 3 & 4 & 5 \\
u_n & 1 & 2 & 4 & 8 & 16 & 32 
\hline
\end{NiceArray}$
\end{enumerate}

However, it’s also possible to use the tools of booktabs: \toprule, \bottomrule and \midrule.

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

\footnote{It’s also possible to use \firsthline with \NiceArray.}
8 The exterior rows and columns

The options \texttt{first-row}, \texttt{last-row}, \texttt{first-col} and \texttt{last-col} allow the composition of exterior rows and columns in the environments of \texttt{nicematrix}.

A potential “first row” (exterior) has the number 0 (and not 1). Idem for the potential “first column”.

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

We have several remarks to do.

- For the environments with an explicit preamble (i.e. \texttt{NiceArray} and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type \texttt{R} for the first column and \texttt{L} for the last one.

- One may wonder how \texttt{nicematrix} determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
  
  - For the environments with explicit preamble, like \texttt{NiceArray} and \texttt{pNiceArray}, the number of columns can obviously be computed from the preamble.
  
  - When the option \texttt{light-syntax} (cf. p. 17) is used, \texttt{nicematrix} has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option \texttt{light-syntax}). The analysis of this whole body gives the number of rows (but not the number of columns).
  
  - In the other cases, \texttt{nicematrix} compute the number of rows and columns during the first compilation and write the result in the \texttt{aux} file for the next run.

However, it’s possible to provide the number of the last row and the number of the last column as values of the options \texttt{last-row} and \texttt{last-col}, tending to an acceleration of the whole compilation of the document. That’s what we will do throughout the rest of the document.

It’s possible to control the appearance of these rows and columns with options \texttt{code-for-first-row}, \texttt{code-for-last-row}, \texttt{code-for-first-col} and \texttt{code-for-last-col}. These options specify tokens that will be inserted before each cell of the corresponding row or column.

\begin{verbatim}
\NiceMatrixOptions{code-for-first-row = \color{red},
  code-for-first-col = \color{blue},
  code-for-last-row = \color{green},
  code-for-last-col = \color{magenta}}
\begin{pNiceArray}[CC|CC][first-row,last-row=5,first-col,last-col=nullify-dots]
  \& C_1 & \Cdots & \& C_4 & \\
10
\end{pNiceArray}
\end{verbatim}
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\ \hline & a_{21} & a_{22} & a_{23} & a_{24} & \hline & a_{31} & a_{32} & a_{33} & a_{34} & & C_1 & C_2 & C_3 & C_4 & L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 & C_1 \end{pNiceMatrix}

Remarks

- As shown in the previous example, an horizontal rule (drawn by \hline) doesn’t extend in the exterior columns and a vertical rule (specified by a “|” in the preamble of the array) doesn’t extend in the exterior rows.\footnote{The latter is not true when the package arydshln is loaded besides nicematrix. In fact, nicematrix and arydshln are not totally compatible because arydshln redefines many internals of array. On another hand, if one really wants a vertical rule running in the first and in the last row, he should use \rule{\textwidth}{1pt} instead of | in the preamble of the array.}

If one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command \OnlyMainNiceMatrix described on page 17.

- A specification of color present in code-for-first-row also applies to a dotted line draw in this exterior “first row” (excepted if a value has been given to xdots/color). Idem for the other exterior rows and columns.

- Logically, the potential option columns-width (described p. 12) doesn’t apply to the “first column” and “last column”.

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

9 The dotted lines to separate rows or columns

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

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

In the environments with an explicit preamble (like \{NiceArray\}, etc.), it’s possible to draw a vertical dotted line with the specifier “:\.

\left(\begin{NiceArray}{CCCC:C}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{NiceArray}\right)
These dotted lines do not extend in the potential exterior rows and columns.

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

It’s possible to change in \texttt{nicematrix} the letter used to specify a vertical dotted line with the option \texttt{letter-for-dotted-lines} available in \texttt{\NiceMatrixOptions}. For example, in this document, we have loaded the package \texttt{arydshln} which uses the letter “::” to specify a vertical dashed line. Thus, by using \texttt{letter-for-dotted-lines}, we can use the vertical lines of both \texttt{arydshln} and \texttt{nicematrix}.

\begin{NiceMatrixOptions}{letter-for-dotted-lines = I} \\
\arrayrulecolor{blue} \\
\left( \begin{NiceArray}{C|C:CIC} \\
  1 & 2 & 3 & 4 & L_1 \\
  5 & 6 & 7 & 8 & L_2 \\
  9 & 10 & 11 & 12 & L_3 \\
  13 & 14 & 15 & 16 & L_4 \\
\end{NiceArray} \right) \\
\arrayrulecolor{black} \\
\right)

We have used the command \texttt{\arrayrulecolor} (\texttt{de\ color\ tbl}) to draw in blue the three rules.

Remark: In the package \texttt{array} (on which the package \texttt{nicematrix} relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule\footnote{In fact, this is true only for \texttt{\hline} and “::” but not for \texttt{\cline}.}. In \texttt{nicematrix}, the dotted lines drawn by \texttt{\hdottedline} and “::” do likewise.

## 10 The width of the columns

In the environments with an explicit preamble (like \texttt{\NiceArray}, \texttt{p\NiceArray}, etc.), it’s possible to fix the width of a given column with the standard letters \texttt{w} and \texttt{W} of the package \texttt{array}. In the environments of \texttt{nicematrix}, the cells of such columns are composed in mathematical mode, whereas, in \texttt{\array} of \texttt{array}, they are composed in text mode.

\begin{NiceMatrix}{wc{1cm}CC} \\
  1 & 12 & -123 \\
  12 & 0 & 0 \\
  4 & 1 & 2 \\
\end{NiceMatrix}

In the environments of \texttt{nicematrix}, it’s also possible to fix the minimal width of all the columns of a matrix directly with the option \texttt{columns-width}.

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

Note that the space inserted between two columns (equal to 2 \texttt{\arraycolsep}) is not suppressed (of course, it’s possible to suppress this space by setting \texttt{\arraycolsep} equal to 0 pt).
It’s possible to give the special value \texttt{auto} to the option \texttt{columns-width}: all the columns of the array will have a width equal to the widest cell of the array.\footnote{The result is achieved with only one compilation (but \LaTeX{} will have written informations in the \texttt{.aux} file and a message requiring a second compilation will appear).}

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

Without surprise, it’s possible to fix the minimal width of the columns of all the matrices of a current scope with the command \texttt{\NiceMatrixOptions}.

\begin{verbatim}
\NiceMatrixOptions{columns-width=10mm}
$\begin{pNiceMatrix}
a & b \\
c & d
\end{pNiceMatrix}$
\end{verbatim}

\begin{verbatim}
\begin{NiceMatrixBlock}[auto-columns-width]
\begin{pNiceMatrix}
a & b \\
c & d
\end{pNiceMatrix}
\end{NiceMatrixBlock}
\end{verbatim}

Several compilations may be necessary to achieve the job.

\section{Block matrices}

This section has no direct link with the previous one where an environment \{\NiceMatrixBlock\} was introduced.

In the environments of \texttt{nicematrix}, it’s possible to use the command \texttt{\Block} in order to place an element in the center of a rectangle of merged cells of the array.

The command \texttt{\Block} must be used in the upper leftmost cell of the array with two arguments. The first argument is the size of the block with the syntax \texttt{i-j} where \texttt{i} is the number of rows of the block and \texttt{j} its number of columns. The second argument is the content of the block (composed in math mode). A Tikz node corresponding to the merged cells is created with the name “\texttt{i-j-block}”. If the user has required the creation of the “medium nodes”, a node of this type is also created with a name suffixed by \texttt{-medium}.

In the following examples, we use the command \texttt{\arrayrulecolor} of \texttt{colortbl}.

\footnote{At this time, this is the only usage of the environment \{\NiceMatrixBlock\} but it may have other usages in the future.}
One may wish to raise the size of the "A" placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like \textbf{large}, \textbf{Large} and \textbf{LARGE}. That's why the command \textbf{Block} provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.

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

## 12 Advanced features

### 12.1 Alignment option in NiceMatrix

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

\begin{NiceMatrix}[L]
\cos x & - \sin x \\
\sin x & \cos x
\end{NiceMatrix}

This is a part of the functionality provided by the environments \texttt{\{pmatrix*\}}, \texttt{\{bmatrix*\}}, etc. of \texttt{mathtools}.

### 12.2 The command \texttt{\rotate}

The package \texttt{nicematrix} provides a command \texttt{\rotate}. When used in the beginning of a cell, this command composes the contents of the cell after a rotation of 90° in the direct sens. In the following command, we use that command in the \texttt{code-for-first-row}.
If the command \rotate is used in the “last row” (exterior to the matrix), the corresponding elements are aligned upwards as shown below.

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

12.3 The option small

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

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

One should note that the environment \{NiceMatrix\} with the option small is not composed \emph{exactly} as the environment \{smallmatrix\}. Indeed, all the environments of nicematrix are constructed upon \{array\} (of the package array) whereas the environment \{smallmatrix\} is constructed directly with an \texttt{\halign} of T\ eX.

In fact, the option small corresponds to the following tuning:
- the cells of the array are composed with \texttt{\scriptstyle};
- \texttt{\arraystretch} is set to 0.47;
- \texttt{\arraycolsep} is set to 1.45 pt;
- the characteristics of the dotted lines are also modified.

12.4 The counters iRow and jCol

In the cells of the array, it’s possible to use the LaTeX counters \texttt{iRow} and \texttt{jCol} which represent the number of the current row and the number of the current column\textsuperscript{23}. Of course, the user must not change the value of these counters which are used internally by nicematrix.

In the \texttt{code-after} (cf. p. 7), \texttt{iRow} represents the total number of rows (excepted the potential exterior rows) and \texttt{jCol} represents the total number of columns (excepted the potential exterior columns).

\textsuperscript{23}We recall that the exterior “first row” (if it exists) has the number 0 and that the exterior “first column” (if it exists) has also the number 0.
If LaTeX counters called \texttt{iRow} and \texttt{jCol} are defined in the document by packages other than \texttt{nicematrix} (or by the user), they are shadowed in the environments of \texttt{nicematrix}.

The package \texttt{nicematrix} also provides commands in order to compose automatically matrices from a general pattern. These commands are \texttt{\pAutoNiceMatrix}, \texttt{\bAutoNiceMatrix}, \texttt{\vAutoNiceMatrix}, \texttt{\VAutoNiceMatrix} and \texttt{\BAutoNiceMatrix}.

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

\[
C = \pAutoNiceMatrix{3-3}{C_{\arabic{iRow},\arabic{jCol}}}
\]

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

12.5 The options \texttt{hlines}, \texttt{vlines} and \texttt{hvlines}

You can add horizontal rules between rows in the environments of \texttt{nicematrix} with the usual command \texttt{\hline} and you can use the specifier "|" to add vertical rules. However, by convenience, the package \texttt{nicematrix} also provides the option \texttt{hlines} (resp. \texttt{vlines}) which will draw all the horizontal (resp. vertical) rules (excepted, of course, the exterior rules corresponding to the exterior rows and columns). The key \texttt{hvlines} is an alias for the conjunction for the keys \texttt{hlines} et \texttt{vlines}.

In the following example, we use the command \texttt{\arrayrulecolor} of \texttt{colortbl}.

\[
\arrayrulecolor{blue}\begin{NiceArray}{|CCCC|}
\hline
 e & e & a & b & c \\
\hline
 e & e & a & b & c \\
 a & a & e & c & b \\
 b & b & c & e & a \\
 c & c & b & a & e
\end{NiceArray}\arrayrulecolor{black}
\]

However, there is a difference between the key \texttt{vlines} and the use of the specifier "|" in the preamble of the environment: the rules drawn by \texttt{vlines} completely cross the double-rules drawn by \texttt{\hline}.

\[
\arrayrulecolor{black}\begin{NiceArray}{|CCCC|}[vlines] \hline
 a & b & c & d \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \end{NiceArray}\hline
\]

For the environments with delimiters (for example \texttt{\pNiceArray} or \texttt{\pNiceMatrix}), the option \texttt{vlines} don’t draw vertical rules on both sides, where are the delimiters (fortunately).
12.6 The option light-syntax

The option \textit{light-syntax}\textsuperscript{24} allows the user to compose the arrays with a lighter syntax, which gives a more readable TeX source.

When this option is used, one should use the semicolon for the end of a row and spaces or tabulations to separate the columns. However, as usual in the TeX world, the spaces after a control sequence are discarded and the elements between curly braces are considered as a whole.

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

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

It’s possible to change the character used to mark the end of rows with the option \texttt{end-of-row}. As said before, the initial value is a semicolon.

When the option \texttt{light-syntax} is used, it is not possible to put verbatim material (for example with the command \texttt{\verb}) in the cells of the array.\textsuperscript{25}

12.7 Use of the column type S of siunitx

If the package \texttt{siunitx} is loaded (before or after \texttt{nicematrix}), it’s possible to use the \texttt{S} column type of \texttt{siunitx} in the environments of \texttt{nicematrix}. The implementation doesn’t use explicitly any private macro of \texttt{siunitx}.

\begin{pNiceArray}[SCw{1\text{cm}}C][nullify-dots,first-row]
\{C_1\} & \& C_n \\
2.3 & 0 & \& \text{\ldots} \\
12.4 & \text{\ldots} & \text{\ldots} \\
1.45 & \text{\ldots} \\
7.2 & 0 & \& \text{\ldots}
\end{pNiceArray}

On the other hand, the d columns of the package dcolumn are not supported by \texttt{nicematrix}.

13 Technical remarks

13.1 Definition of new column types

The package \texttt{nicematrix} provides the command \texttt{\OnlyMainNiceMatrix} which is meant to be used in definitions of new column types. Its argument is evaluated if and only if we are in the main part of the array, that is to say not in an eventual exterior row.

For example, one may wish to define a new column type ? in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job:\textsuperscript{26}:

\begin{verbatim}
\begin{NiceMatrix}
\{C_1\} & \& C_n \\
2.3 & 0 & \& \text{\ldots} \\
12.4 & \text{\ldots} & \text{\ldots} \\
1.45 & \text{\ldots} \\
7.2 & 0 & \& \text{\ldots}
\end{NiceMatrix}
\end{verbatim}

\textsuperscript{24}This option is inspired by the package \texttt{spalign} of Joseph Rabinoff.

\textsuperscript{25}The reason is that, when the option \texttt{light-syntax} is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn’t behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.

\textsuperscript{26}The command \texttt{\vrule} is a TeX (and not LaTeX) command.
The heavy vertical rule won’t extend in the exterior rows:
\begin{pNiceArray}{CC?CC}[first-row,last-row=3]
\begin{NiceMatrix}
C_1 & C_2 & C_3 & C_4 \\
\begin{array}{cccc}
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{array}
\end{NiceMatrix}
\end{pNiceArray}

The specifier \texttt{?} may be used in a standard environment \{array\} (of the package \texttt{array}) and, in this case, the command \texttt{\OnlyMainNiceMatrix} is no-op.

\section{The names of the PGF nodes created by \texttt{nicematrix}}

We have said that, when a name is given to an environment of \texttt{nicematrix}, it’s possible to access the PGF/Tikz nodes through this name (cf. p. 6).

That’s the recommended way to access these nodes. However, we describe now the internal names of these nodes.

The environments created by \texttt{nicematrix} are numbered by an internal global counter. The command \texttt{\NiceMatrixLastEnv} provides the number of the last environment of \texttt{nicematrix} (for \LaTeX, it’s a “fully expandable” command and not a counter).

For the environment of number \texttt{n}, the node in row \texttt{i} and column \texttt{j} has the name \texttt{nm-n-i-j}. The \texttt{medium} and \texttt{large} nodes have the same name, suffixed by \texttt{-medium} and \texttt{-large}.

\subsection{Diagonal lines}

By default, all the diagonal lines\footnote{We speak of the lines created by \texttt{\Ddots} and not the lines created by a command \texttt{\line} in code-after.} 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 \texttt{\Ddots} in the array can have a marked effect on the final result.

In the following examples, the first \texttt{\Ddots} instruction is written in color:

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

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

It’s possible to turn off the parallelization with the option \texttt{parallelize-diags} set to \texttt{false}:

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

The same example without parallelization:
13.4 The “empty” cells

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

For nicematrix, the precise rules are as follow.

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

\begin{pmatrix}
ad & b \\
c & \\
\end{pmatrix}

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

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

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

13.5 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. The environment \{matrix\} of amsmath and its variants (\{pmatrix\}, \{vmatrix\}, etc.) of amsmath prefer to delete these spaces with explicit instructions \hskip -\arraycolsep. The package nicematrix does the same in all its environments, \{NiceArray\} included. However, if the user wants the environment \{NiceArray\} behaving by default like the environment \{array\} of array (for example, when adapting an existing document) it’s possible to control this behaviour with the option exterior-arraycolsep, set by the command \NiceMatrixOptions. With this option, exterior spaces of length \arraycolsep will be inserted in the environments \{NiceArray\} (the other environments of nicematrix are not affected).

13.6 The class option draft

When the class option draft is used, the dotted lines are not drawn, for a faster compilation.

13.7 A technical problem with the argument of \\\n
For technical, reasons, if you use the optional argument of the command \\\, the vertical space added will also be added to the “normal” node corresponding at the previous node.

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

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

---

28 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).

29 And not by inserting @{} on both sides of the preamble of the array. As a consequence, the length of the \hline is not modified and may appear too long, in particular when using square brackets.
\begin{pNiceMatrix}
\begin{pmatrix}
a & \frac{AB}{77} \\
\end{pmatrix}
\end{pNiceMatrix}

The other solution is to use the command \texttt{\multicolumn} in the previous cell.

\begin{pNiceMatrix}
\begin{pmatrix}
a & \multicolumn{1}{c}{\frac{AB}{77}} \\
\end{pmatrix}
\end{pNiceMatrix}

### 13.8 Obsolete environments

The version 3.0 of \texttt{nicematrix} has introduced the environment \texttt{\{pNiceArray\}} (and its variants) with the options \texttt{first-row}, \texttt{last-row}, \texttt{first-col} and \texttt{last-col}.

Consequently the following environments present in previous versions of \texttt{nicematrix} are deprecated:

- \texttt{\{NiceArrayCwithDelims\}};
- \texttt{\{pNiceArrayC\}}, \texttt{\{bNiceArrayC\}}, \texttt{\{BNiceArrayC\}}, \texttt{\{vNiceArrayC\}}, \texttt{\{VNiceArrayC\}};
- \texttt{\{NiceArrayRCwithDelims\}};
- \texttt{\{pNiceArrayRC\}}, \texttt{\{bNiceArrayRC\}}, \texttt{\{BNiceArrayRC\}}, \texttt{\{vNiceArrayRC\}}, \texttt{\{VNiceArrayRC\}}.

Since the version 3.12, the only way to use these environments is loading \texttt{nicematrix} with the option \texttt{obsolete-environments}.

However, these environments will certainly be completely deleted in a future version of \texttt{nicematrix}.

### 14 Examples

#### 14.1 Dotted lines

A permutation matrix (as an example, we have raised the value of \texttt{xdots/shorten}).

\begin{pNiceMatrix}[xdots/shorten=0.6em]
0 & 1 & 0 & \cdots & 0 \\
\vdots & & & \ddots & \\
0 & 0 & & 1 & \\
1 & 0 & & \cdots & 0
\end{pNiceMatrix}

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

\begin{pNiceMatrix}[xdots/shorten=0.9em]
1 & \cdots & 1 \\
\vdots & & \vdots \\
1 & 0 & 0
\end{pNiceMatrix}
An example with `\multicolumn`:

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

An example with `\Hdotsfor`:

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

An example for the resultant of two polynomials:

\begin{vNiceArray}{CCCC:CCC}[columns-width=6mm]
\setlength{\extrarowheight}{1mm}
a_0 & && & b_0 & & \\
a_1 & \Ddots && & b_1 & \Ddots & \\
\Vdots & \Ddots && & \Vdots & \Ddots & b_0 \\
a_p & && a_0 & & & b_1 \\
& \Ddots && a_1 & b_q & & \Vdots \\
& && \Vdots & & \Ddots & \\
& && a_p & & & b_q
\end{vNiceArray}
An example for a linear system (the vertical rule has been drawn in blue with the tools of\color{blue}):

\begin{NiceArray}{*6C|C}
\nullify-dots, last-col, code-for-last-col=\scriptstyle
\begin{array}{cccccccccc}
1 & 1 & 1 & \cdots & & 1 & 0 & \\
0 & 0 & 1 & \cdots & & 0 & & L_2 \gets L_2 - L_1 \\
0 & 0 & 1 & \cdots & & \vdots & & L_3 \gets L_3 - L_1 \\
\vdots & \vdots & \vdots & \cdots & & 0 & \\
0 & \vdots & \vdots & \vdots & \cdots & 0 & 1 & L_n \gets L_n - L_1 \\
\end{array}
\end{NiceArray}

\begin{pNiceMatrix}[first-row, first-col] & x & \cdots & x & 0 \\
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
0 & \vdots & \vdots & \cdots & 0 \\
\end{pNiceMatrix}

\subsection*{14.2 Dotted lines which are no longer dotted}

The option line-style controls the style of the lines drawn by \Ldots, \Cdots, etc. Thus, it’s possible with these commands to draw lines which are not longer dotted.

\NiceMatrixOptions
\nullify-dots, code-for-first-col = \color{blue}, code-for-first-col=\color{blue}
\begin{pNiceMatrix}[first-row, first-col]
& 1 & 1 & \ldots & 1 & 0 \\
0 & 1 & 0 & \cdots & 0 & \vdots & L_2 \gets L_2 - L_1 \\
0 & 0 & 1 & \cdots & 0 & \vdots & L_3 \gets L_3 - L_1 \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
0 & \vdots & \vdots & \cdots & 0 & \vdots & L_n \gets L_n - L_1 \\
\end{pNiceMatrix}

\subsection*{14.3 Width of the columns}

In the following example, we use \NiceMatrixBlock with the option auto-columns-width because we want the same automatic width for all the columns of the matrices.
14.4 How to highlight cells of the matrix

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

\usepackage{tikz}
\usetikzlibrary{fit}

In order to highlight a cell of a matrix, it’s possible to “draw” one of the correspondant nodes (the “normal node”, the “medium node” or the “large node”). In the following example, we use the “large nodes” of the diagonal of the matrix (with the Tikz key “name suffix”, it’s easy to use the “large nodes”).

We redraw the nodes with other nodes by using the Tikz library fit. Since we want to redraw the nodes exactly, we have to set \texttt{inner sep} = 0 pt (if we don’t do that, the new nodes will be larger that the nodes created by \texttt{nicematrix}).

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

\begin{verbatim}
1 1 1 1
2 4 8 16
3 9 27 81
4 16 64 256
\end{verbatim}

\begin{verbatim}
\begin{pNiceArray}{CCCC:C}[create-large-nodes,margin,extra-margin = 2pt]
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\CodeAfter
\end{pNiceArray}
\end{verbatim}
We should remark that the rules we have drawn are drawn after the construction of the array and thus, they don’t spread the cells of the array. We recall that, on the other side, the command \hline, the specifier “|” and the options hlines and vlines spread the cells (when the package array is loaded but, when the package nicematrix is loaded, array is always loaded).  

The package nicematrix is constructed upon the environment \{array\} and, therefore, it’s possible to use the package colortbl in the environments of nicematrix. However, it’s not always easy to do a fine tuning of colortbl. That’s why we propose another method to highlight a row of the matrix. We create a rectangular Tikz node which encompasses the nodes of the second row with the Tikz library fit. This Tikz node is filled after the construction of the matrix. In order to see the text under this node, we have to use transparency with the blend mode equal to multiply.

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

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

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

$\begin{bNiceMatrix} 
0 \ldots \ldots 0 \\
1 \ldots \ldots 1 \\
0 \ldots \ldots 0 
\end{bNiceMatrix}$

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

\[\begin{pmatrix} 
0 & 0 \\
1 & 1 \\
0 & 0 
\end{pmatrix}\]
We recall that, for a rectangle of merged cells (with the command \Block), a Tikz node is created for the set of merged cells with the name \textit{i-j-block} where \textit{i} and \textit{j} are the number of the row and the number of the column of the upper left cell (where the command \Block has been issued). If the user has required the creation of the medium nodes, a node of this type is also created with a name suffixed by \textit{-medium}.

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

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

Consider now the following matrix which we have named \textit{example}.

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

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

\begin{tikzpicture}[mes-options/.style={remember picture, overlay, name prefix = exemple-}, highlight/.style = {fill = red!15, blend mode = multiply, inner sep = 0pt, fit = #1}]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}

We obtain the following matrix.

\begin{pNiceMatrix}[name suffix = -medium]
a & a + b & a + b + c & L_1 \\
a & a & a + b & L_2 \\
a & a & a & L_3
\CodeAfter
\tikz \node \[highlight = (1-1) (1-3)] {} ;
\end{pNiceMatrix}

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

\begin{tikzpicture}[mes-options, name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
We obtain the following matrix.

\[
\begin{pmatrix}
  a & a+b & a+b+c \\ \\
  a & a & a+b \\ \\
  a & a & a \\
\end{pmatrix}
\]

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

\[
\begin{pNiceArray}{>{\strut}CCCC}
\text{A}_{11} & \text{A}_{12} & \text{A}_{13} & \text{A}_{14} \\
\text{A}_{21} & \text{A}_{22} & \text{A}_{23} & \text{A}_{24} \\
\text{A}_{31} & \text{A}_{32} & \text{A}_{33} & \text{A}_{34} \\
\text{A}_{41} & \text{A}_{42} & \text{A}_{43} & \text{A}_{44}
\end{pNiceArray}
\]

14.5 Direct use of the Tikz nodes

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

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

\[
\begin{NiceMatrixBlock}\texttt{[auto-columns-width]}
\NiceMatrixOptions{nullify-dots}
\begin{array}{cc}
& \\
\text{The matrix B has a “first row” (for C_j) and that’s why we use the key \texttt{first-row}.}
\begin{bNiceArray}{C>{\strut}CCCC}\texttt{[name=B,first-row]}
& & C_j \\
\text{b}_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
\Vdots & & \Vdots & & \Vdots \\
& & b_{kj} \\
& & \Vdots \\
\text{b}_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn}
\end{bNiceArray} \\
\text{The matrix A has a “first column” (for L_i) and that’s why we use the key \texttt{first-col}.}
\end{NiceMatrixBlock}
\]
In the matrix product, the two dotted lines have an open extremity.

\begin{bNiceArray}{CC>{\strut}CCC}
  \begin{array}{c}
  \end{array}
\end{bNiceArray}

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

15 Implementation

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

However, when the option renew-dots is used, the commands \cdots, \ldots, \ldots, \vdots, \ddots and \iddots are redefined in the environments provided by nicematrix as explained previously.

In the same way, if the option renew-matrix is used, the environment \{matrix\} of amsmath is redefined.

On the other hand, the environment \{array\} is never redefined.

Of course, the package nicematrix uses the features of the package array. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent: the package nicematrix relies upon the fact that the package \{array\} uses \ialign to begin the \halign.
Declaration of the package and packages loaded

The prefix nicematrix has been registered for this package.
See: http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf
<@@nicematrix>

First, we load pgfcore and the module shapes. We do so because it’s not possible to use \usepgfmodule in \ExplSyntaxOn.

\RequirePackage{pgfcore}
\usepgfmodule{shapes}

We give the traditional declaration of a package written with expl3:
\RequirePackage{l3keys2e}
\ProvidesExplPackage{nicematrix}{\myfiledate}{\myfileversion}{Mathematical matrices with PGF/TikZ}

The version of 2020/02/08 of expl3 has replaced \l_keys_key_tl by \l_keys_key_str. We have immediately changed in this file. Now, you test the existence of \l_keys_key_str in order to detect whether the version of LaTeX used by the final user is up to date.
\msg_new:nnn { nicematrix } { expl3-too-old }
{ Your version of LaTeX (especially expl3) is too old.
You can go on but you will probably have other errors
if you use the functionalities of nicematrix. }
\cs_if_exist:NF \l_keys_key_str { \msg_error:nn { nicematrix } { expl3-too-old } }

We test the class option draft. In this case, we raise the flag \c_@@_draft_bool because we won’t draw the dotted lines if the option draft is used.
\bool_new:N \c_@@_draft_bool
\DeclareOption { draft } { \bool_set_true:N \c_@@_draft_bool }
\DeclareOption* { }
\ProcessOptions \relax

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

We load some packages.
\RequirePackage { array }
\RequirePackage { amsmath }
\RequirePackage { xparse }
\cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
\cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
\cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_redirect_name:nn { \msg_redirect_name:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_redirect_name:nnn { \msg_redirect_name:nnnn { nicematrix } }

Technical definitions

In some constructions, we will have to use a `{pgfpicture}` which must be replaced by a `{tikzpicture}` if Tikz is loaded. However, this switch between `{pgfpicture}` and `{tikzpicture}` can’t be done dynamically with a conditional because, when the Tikz library `external` is loaded by the user, the pair `{tikzpicture}`-{end{tikzpicture} must be statically “visible” (even when externalization is not activated). That’s why we create `{c_@@_pgfortikzpicture_tl}` and `{c_@@_endpgfortikzpicture_tl}` which will be used to construct in a `{AtBeginDocument}` the correct version of some commands.

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

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

This definition is a variant of the standard definition of `{ddots}`.

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

29
We also define a counter to count the environments \{NiceMatrixBlock\}.

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

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

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

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

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

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

The length \l_{\@@}_xdots_shorten_dim is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).
The length \l_{@@}\_radius\_dim is the radius of the dots for the dotted lines (for \hdottedline and \dottedline and for all the other dotted lines when line-style is equal to standard, which is the initial value). The initial value is 0.53 pt but it will be changed if the option small is used (to 0.37 pt).

The name of the current environment or the current command (despite the name which contains env).

The string \g_{@@}\_com\_or\_env\_str will contain the word command or environment whether we are in a command of nicematrix or a an environment of nicematrix. The default value is environment.

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

The counters \l_{@@}\_save\_iRow\_int and \l_{@@}\_save\_jCol\_int will be used to save the values of the eventual LaTeX counters iRow and jCol. These LaTeX counters will be restored at the end of the environment.

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

The token list \l_{@@}\_xdots\_line\_style\_tl corresponds to the option tikz of the commands \Cdots, \Ldots, etc. and of the options line-style for the environments and \NiceMatrixOptions. The constant \c_{@@}\_standard\_tl will be used in some tests.
Variables for the exterior rows and columns

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

- First row
  The integer \l_@@_first_row_int is the number of the first row of the array. The default value is 1, but, if the option first-row is used, the value will be 0. As usual, the global version is for the passage in the \group_insert_after:N.

\begin{verbatim}
\int_new:N \l_@@_first_row_int
\int_set:Nn \l_@@_first_row_int 1
\end{verbatim}

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

\begin{verbatim}
\int_new:N \l_@@_first_col_int
\int_set:Nn \l_@@_first_col_int 1
\end{verbatim}

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

\begin{verbatim}
\int_new:N \l_@@_last_row_int
\int_set:Nn \l_@@_last_row_int { -2 }
\end{verbatim}

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

\begin{verbatim}
\bool_new:N \l_@@_last_row_without_value_bool
\end{verbatim}

Idem for \l_@@_last_col_without_value_bool

\begin{verbatim}
\bool_new:N \l_@@_last_col_without_value_bool
\end{verbatim}

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

\begin{verbatim}
\int_new:N \l_@@_last_col_int
\int_set:Nn \l_@@_last_col_int { -2 }
\end{verbatim}

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

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

\begin{verbatim}
\begin{pNiceArray}{CC}
1 & 2 \\
3 & 4
\end{pNiceArray}
\end{verbatim}

\[31\]We can’t use \l_@@_last_row_int for this usage because, if nicematrix has read its value from the aux file, the value of the counter won’t be \(-1\) any longer.
In such a code, the “last column” specified by the key `last-col` is not used. We want to be able to detect such a situation and we create a boolean for that job.

\bool_new:N \g_@@_last_col_found_bool

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

The column \texttt{S} of \texttt{siunitx}

We want to know whether the package \texttt{siunitx} is loaded and, if it is loaded, we redefine the \texttt{S} columns of \texttt{siunitx}.

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

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

\renewcommand*{\NC@rewrite@S}[1][]
{\@temptokena \exp_after:wN
 { \tex_the:D \@temptokena > { \_\_siunitx_table_collect_begin: S {#1} } c
 < { \_\_siunitx_table_print: } }
 \NC@find}

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

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

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

\cs_set_protected:Npn \@@_adapt_S_column:
{\bool_if:NT \c_@@_siunitx_loaded_bool
 { \group_begin:
 \@temptokena = { }
We protect \NC@find which is at the end of \NC@rewriteS.
\cs_set_eq:NN \NC@find \prg_do_nothing:
\NC@rewriteS { }

Conversion of the toks \@temptokena in a token list of expl3 (the toks are not supported by expl3 but we can, nevertheless, use the option V for \tl_gset:NV).
\tl_gset:NV \g_tmpa_tl \@temptokena
\group_end:
\tl_new:N \c_@@_table_collect_begin_tl
\tl_set:Nx \l_tmpa_tl { \tl_item:Nn \g_tmpa_tl 2 }
\tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \l_tmpa_tl 1 }
\tl_new:N \c_@@_table_print_tl
\tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl { -1 } }
The token lists \c_@@_table_collect_begin_tl and \c_@@_table_print_tl contain now the two commands of siunitx.

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

The command \@@_renew_NC@rewriteS: will be used in each environment of nicematrix in order to “rewrite” the S column in each environment (only if the boolean \c_@@_siunitx_loaded_bool is raised, of course).
\cs_new_protected:Npn \@@_renew_NC@rewriteS:
\begin { pgfscope }
\pgfset {
outer~sep = \c_zero_dim ,
inner~sep = \c_zero_dim ,
minimum~size = \c_zero_dim
}
\pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
\pgfnode

The following command is only for efficiency. It must not be protected because it will be used (for instance) in names of PGF nodes.
\cs_new:Npn \@@_succ:n #1 { \the \numexpr #1 + 1 \relax }
\cs_new:Npn \@@_pred:n #1 { \the \numexpr #1 - 1 \relax }

Command for creation of rectangle nodes
The following command should be used in a \{pgfpicture\}. It creates an rectangular (empty but with a name) when the four corners are given.
#1 is the name of the node which will be created; #2 and #3 are the coordinates of one of the corner of the rectangle; #4 and #5 are the coordinates of the opposite corner.
\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
\begin { pgfscope }
\pgfset {
outer~sep = \c_zero_dim ,
ininner~sep = \c_zero_dim ,
minimum~size = \c_zero_dim
}
\pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
\pgfnode
The command \@@_pgf_rect_node:nnn is a variant of \@@_pgr_rect_node:nnnn: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

\begin{pgfscope}
\pgfset{
\outersep = \c_zero_dim ,
\innersep = \c_zero_dim ,
\minimumsize = \c_zero_dim
}
\pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
\pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
\pgfnode{rectangle}{center}{\vbox_to_ht:nn{\dim_abs:n \l_tmpb_dim}\vfill\hbox_to_wd:nn{\dim_abs:n \l_tmpa_dim}{}}{#1}{}
\end{pgfscope}

The boolean \l@@light_syntax_bool corresponds to the option light-syntax.
\bool_new:N \l@@light_syntax_bool

The token list \l@@baseline_str may contain one of the three values t, c or b as in the option of the environment \{array\}. However, it may also contain an integer (which represents the number of the row to which align the array).
\str_new:N \l@@baseline_str
\str_set:Nn \l@@baseline_str c

The flag \l@@exterior_arraycolsep_bool corresponds to the option exterior-arraycolsep. If this option is set, a space equal to \arraycolsep will be put on both sides of an environment \{NiceArray\} (as it is done in \{array\} of array).
\bool_new:N \l@@exterior_arraycolsep_bool
The flag \l_@@_parallelize_diags_bool controls whether the diagonals are parallelized. The initial value is true.
\begin{verbatim}
\bool_new:N \l_@@_parallelize_diags_bool
\bool_set_true:N \l_@@_parallelize_diags_bool
\end{verbatim}

The flag \l_@@_hlines_bool corresponds to the option \hlines and the flag \l_@@_vlines_bool to the option \vlines.
\begin{verbatim}
\bool_new:N \l_@@_hlines_bool
\bool_new:N \l_@@_vlines_bool
\end{verbatim}

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.).
\begin{verbatim}
\bool_new:N \l_@@_nullify_dots_bool
\end{verbatim}

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).
\begin{verbatim}
\bool_new:N \l_@@_auto_columns_width_bool
\end{verbatim}

The token list \l_@@_name_str will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.
\begin{verbatim}
\str_new:N \l_@@_name_str
\end{verbatim}

The boolean \l_@@_medium_nodes_bool will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.
\begin{verbatim}
\bool_new:N \l_@@_medium_nodes_bool
\bool_new:N \l_@@_large_nodes_bool
\end{verbatim}

The dimension \l_@@_left_margin_dim correspond to the option left-margin. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (\hdottedline).
\begin{verbatim}
\dim_new:N \l_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim
\end{verbatim}

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.
\begin{verbatim}
\dim_new:N \g_@@_width_last_col_dim
\dim_new:N \g_@@_width_first_col_dim
\end{verbatim}

The dimensions \l_@@_extra_left_margin_dim and \l_@@_extra_right_margin_dim correspond to the options extra-left-margin and extra-right-margin.
\begin{verbatim}
\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim
\end{verbatim}

The token list \l_@@_end_of_row_tl corresponds to the option end-of-row. It specifies the symbol used to mark the ends of rows when the light syntax is used.
\begin{verbatim}
\tl_new:N \l_@@_end_of_row_tl
\tl_set:Nn \l_@@_end_of_row_tl { ; }
\end{verbatim}

The following parameter is for the color the dotted lines drawn by \Cdots, \Ldots, \Vdots, \Ddots, \Iddots and \Hdots for but not the dotted lines drawn by \hdottedline and “:”. 
\begin{verbatim}
\tl_new:N \l_@@_xdots_color_tl
\end{verbatim}
Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option \texttt{columns-width} or the option \texttt{auto-columns-width} of the environment \texttt{NiceMatrixBlock}). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is function of its size. That’s why we create an option called \texttt{max-delimiter-width} which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

\begin{verbatim}
\bool_new:N \l_@@_max_delimiter_width_bool
\end{verbatim}

First, we define a set of keys “\texttt{NiceMatrix / Global}” which will be used (with the mechanism of \texttt{.inherit:n}) by other sets of keys.

\begin{verbatim}
\keys_define:nn { NiceMatrix / Global } {
  xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
  max-delimiter-width .bool_set:N = \l_@@_max_delimiter_width_bool ,
  light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
  light-syntax .default:n = true ,
  end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
  end-of-row .value_required:n = true ,
  code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
  code-for-first-col .value_required:n = true ,
  code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
  code-for-last-col .value_required:n = true ,
  code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
  code-for-first-row .value_required:n = true ,
  code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
  code-for-last-row .value_required:n = true ,
  small .bool_set:N = \l_@@_small_bool ,
  hlines .bool_set:N = \l_@@_hlines_bool ,
  vlines .bool_set:N = \l_@@_vlines_bool ,
  hlines .meta:n = \{ hlines , vlines \} ,
  parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
}
\end{verbatim}

With the option \texttt{renew-dots}, the command \texttt{\ldots}, \texttt{\vldots}, \texttt{\Vdots} and \texttt{\Ddots} are redefined and behave like the commands \texttt{\Cdots}, \texttt{\Ldots}, \texttt{\Vdots} and \texttt{\Ddots}.

\begin{verbatim}
\renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
\end{verbatim}
In some circumstances, the "medium nodes" are created automatically, for example when a dotted line has an "open" extremity (ident for the "large nodes").

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

\begin{verbatim}
\keys_define:nn { NiceMatrix / Env }
{ }
\end{verbatim}

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

We test whether we are in the measuring phase of an environment of \texttt{amsmath} (always loaded by \texttt{nicematrix}) because we want to avoid a fallacious message of duplicate name in this case.
We begin the construction of the major sets of keys (used by the different user commands and
environments).

\keys_define:nn { NiceMatrix }
{
    NiceMatrixOptions .inherit:n =
    {
        NiceMatrix / Global ,
    },
    NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
    NiceMatrix .inherit:n =
    {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
    },
    NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
    NiceArray .inherit:n =
    {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
    },
    NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
    pNiceArray .inherit:n =
    {
        NiceMatrix / Global ,
        NiceMatrix / Env ,
    },
    pNiceArray / xdots .inherit:n = NiceMatrix / xdots
}

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

\keys_define:nn { NiceMatrix / NiceMatrixOptions }
{
With the option renew-matrix, the environment \{matrix\} of amsmath and its variants are redefined
to behave like the environment \{NiceMatrix\} and its variants.
renew-matrix .code:n = \@@_renew_matrix: ,
renew-matrix .value_forbidden:n = true ,
transparent .meta:n = \{ renew-dots , renew-matrix \} ,
transparent .value_forbidden:n = true,
The option exterior-arraycolsep will have effect only in \{NiceArray\} for those who want to have
for \{NiceArray\} the same behaviour as \{array\}.

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

If the option columns-width is used, all the columns will have the same width.
In \NiceMatrixOptions, the special value auto is not available.
columns-width .code:n =
\str_if_eq:nnTF { #1 } { auto }
{ \@@_error:n { Option-auto-for-columns-width } }
{ \dim_set:Nn \l_@@_columns_width_dim { #1 } },

Usually, an error is raised when the user tries to give the same to name two distincts environments
of nicematrix (theses names are global and not local to the current TeX scope). However, the option
allow-duplicate-names disables this feature.
allow-duplicate-names .code:n =
\@@_msg_redirect_name:nn { Duplicate-name } { none } ,
allow-duplicate-names .value_forbidden:n = true ,
By default, the specifier used in the preamble of the array (for example in \{pNiceArray\}) to draw a vertical dotted line between two columns is the colon “:”. However, it’s possible to change this letter with letter-for-dotted-lines and, by the way, the letter “:” will remain free for other packages (for example arydshln).

\texttt{letter-for-dotted-lines .code:n =}
\begin{verbatim}
  \int_compare:nTF { \tl_count:n { #1 } = 1 }
  { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
  { \@@_error:n { Bad-value-for-letter-for-dotted-lines } }
\end{verbatim}

\texttt{letter-for-dotted-lines .value_required:n = true ,}
\texttt{unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }}
\begin{verbatim}
\str_new:N \l_@@_letter_for_dotted_lines_str \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
\end{verbatim}

\texttt{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.
\begin{verbatim}
\NewDocumentCommand \NiceMatrixOptions { m } { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }
\end{verbatim}

We finalise the definition of the set of keys “NiceMatrix / NiceMatrix” with the options specific to \{NiceMatrix\}.
\begin{verbatim}
\keys_define:nn { NiceMatrix / NiceMatrix } { \int_if_empty:nTF {#1} { \bool_set_true:N \l_@@_last_col_without_value_bool \int_set:Nn \l_@@_last_col_int { -1 } } { \int_set:Nn \l_@@_last_col_int { #1 } } , \l .code:n = \tl_set:Nn \l_@@_type_of_col_tl L , \r .code:n = \tl_set:Nn \l_@@_type_of_col_tl R , \L .code:n = \tl_set:Nn \l_@@_type_of_col_tl L , \R .code:n = \tl_set:Nn \l_@@_type_of_col_tl R , unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix } }
\end{verbatim}

We finalise the definition of the set of keys “NiceMatrix / NiceArray” with the options specific to \{NiceArray\}.
\begin{verbatim}
\keys_define:nn { NiceMatrix / NiceArray } { \int_if_empty:nF {#1} { \@@_error:n { last-col-non-empty-for-NiceArray } } \int_zero:N \l_@@_last_col_int , unknown .code:n = \@@_error:n { Unknown-option-for-NiceArray } }
\end{verbatim}

In the environments \{NiceArray\} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.
\begin{verbatim}
\int_if_empty:nF {#1} { \@@_error:n { last-col-non-empty-for-NiceArray } } \int_zero:N \l_@@_last_col_int , unknown .code:n = \@@_error:n { Unknown-option-for-NiceArray }
\end{verbatim}

We finalise the definition of the set of keys “NiceMatrix / pNiceArray” with the options specific to \{pNiceArray\}.
\begin{verbatim}
\keys_define:nn { NiceMatrix / pNiceArray } { \int_zero:N \l_@@_first_col_int , \int_if_empty:nF {#1} { \@@_error:n { last-col-non-empty-for-NiceArray } } \int_zero:N \l_@@_last_col_int , \int_zero:N \l_@@_first_row_int , \int_zero:N \l_@@_last_row_int , \l_@@_last_row_int.default:n = -1 , unknown .code:n = \@@_error:n { Unknown-option-for-NiceMatrix }
\end{verbatim}
Important code used by \texttt{NiceArrayWithDelims}

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

\begin{verbatim}
\cs_new_protected:Npn \@@ Cell:
\{
    \int_gincr:N \c@jCol

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

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

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

    The content of the cell is composed in the box \texttt{l_@@ cell_box} because we want to compute some dimensions of the box. The \texttt{\hbox_set_end:} corresponding to this \texttt{\hbox_set:Nw} will be in the \texttt{\@@ end Cell:} (and the \texttt{c_math_toggle_token} also).

    \hbox_set:Nw \l_@@ cell_box
    \c_math_toggle_token
    \bool_if:NT \l_@@ small_bool \scriptstyle

    We will call \texttt{corners} of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn’t always exist simultaneously).

    The codes \texttt{l_@@ code_for_first_row_tl} and \texttt{al} don’t apply in the corners of the matrix.

    \int_compare:nNnTF \c@iRow = 0
    { \int_compare:nNnT \c@jCol > 0
        { \l_@@ code_for_first_row_tl
            \xglobal \colorlet { nicematrix-first-row } { . } }
        }
    }
    { \int_compare:nNnT \c@iRow = \l_@@ last_row_int
        { \l_@@ code_for_last_row_tl
            \xglobal \colorlet { nicematrix-last-row } { . } }
        }
    }
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@ begin_of_row:
\{
    \int_gincr:N \c@iRow
    \dim_gset_eq:NN \g_@@ dp_ante_last_row_dim \g_@@ dp_last_row_dim
    \dim_gset:Nn \g_@@ dp_last_row_dim { \box_dp:N \arstrutbox }
    \dim_gset:Nn \g_@@ ht_last_row_dim { \box_ht:N \arstrutbox }
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgfcoordinate
    \str_if_empty:NF \l_@@ name_str
    { \pgfnodealias
        { \@@ env: - row - \int_use:N \c@iRow - base }
    }
\end{verbatim}
The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the last-row, some lines will be dynamically added to this command.

\cs_new_protected:Npn \@@_update_for_first_and_last_row: 
\{
    \int_compare:nNnTF \c@iRow = 0 
    \{
        \dim_gset:Nn \g_@@_dp_row_zero_dim 
        \{
            \dim_max:nn \g_@@_dp_row_zero_dim \{ \box_dp:N \l_@@_cell_box \} 
        \}
        \dim_gset:Nn \g_@@_ht_row_zero_dim 
        \{
            \dim_max:nn \g_@@_ht_row_zero_dim \{ \box_ht:N \l_@@_cell_box \} 
        \}
    \}
    \{
        \int_compare:nNnT \c@iRow = 1 
        \{
            \dim_gset:Nn \g_@@_ht_row_one_dim 
            \{
                \dim_max:nn \g_@@_ht_row_one_dim \{ \box_ht:N \l_@@_cell_box \} 
            \}
        \}
    \}
\}
\cs_new_protected:Npn \@@_end_Cell: 
\{
    \c_math_toggle_token
    \hbox_set_end:
\}

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

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

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

\@@_update_for_first_and_last_row:

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

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

However, it’s very difficult to determine whether a cell is empty. As of now, we use the following technic:

- if the width of the box \l_@@_cell_box (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have use a \rlap, a \llap or a \mathclap of mathtools.
- the cells with a command \ldots or \cdots, \Vdots, etc., should also be considered as empty; if \nullify-dots is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of code-after); however, if \nullify-dots is not in force, a phantom of \ldots, \cdots, \vdots is inserted and its width is not equal to zero; that’s why these commands raise a boolean \g_@@_empty_cell_bool and we begin by testing this boolean.

\bool_if:NTF \g_@@_empty_cell_bool 
\{
    \box_use_drop:N \l_@@_cell_box
    \bool_gset_false:N \g_@@_empty_cell_bool
\}

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

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

The first argument of the following command \@@_instruction_of_type:nn defined below is the type of the instruction (\texttt{Cdots}, \texttt{Vdots}, \texttt{Ddots}, etc.). The second argument is the list of options. This command writes in the corresponding \g_@@_type_lines_tl the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,
\begin{pNiceMatrix}
  1 & 2 & 3 & 4 \\[-2.2ex]
  5 & \Cdots & & 6 \\[-2.2ex]
  7 & \Cdots[\color=red] \\[-2.2ex]
\end{pNiceMatrix}

the content of \g_@@_Cdots_lines_tl will be:
\@@_draw_Cdots:nnn {2}{2}{} \@@_draw_Cdots:nnn {3}{2}{\color=red}

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

\bool_if:NTF \c_@@_draft_bool
  { \cs_set_protected:Npn \@@_instruction_of_type:nn \#1 \#2 { } }
\begin{pNiceMatrix}
  1 & 2 & 3 & 4 \\
  5 & \Cdots & & 6 \\
  7 & \Cdots[\color=red] \\
\end{pNiceMatrix}
It’s important to use a \texttt{tl_gput_right:cx} and not a \texttt{tl_gput_left:cx} because we want the \texttt{\textbackslash \textdottedash} lines to be drawn in the order of appearance in the array (for parallelisation).

\begin{verbatim}
\tl_gput_right:cx
{} \_g_@@_ #1 _ lines _ tl 
{} \use:c { \_g_@@_ draw _ #1 : nnn }
{} \int_use:N \c@iRow 
{} \int_use:N \c@jCol 
{} \exp_not:n { #2 } 
}
\end{verbatim}

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

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

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

\begin{verbatim}
[ \str_if_eq:VnTF \l_@@_baseline_str c c t ]
\end{verbatim}

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

\begin{verbatim}
\cs_set_eq:NN \@@_standard_ialign: \ialign
\end{verbatim}

The following must \textit{not} be protected because it begins with \texttt{\textbackslash noalign}.

\begin{verbatim}
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i:
{
  \int_gzero:N \c@jCol
  \hbox
  { \pgfpicture
    \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
    \pgfpointorigin
    \str_if_empty:NF \l_@@_name_str
      { \pgfnodealias
          { \@@_env: - row - \int_use:N \c@iRow - row }
          { \l_@@_name_str - row - \int_use:N \c@iRow - row }
      }
    \endpgfpicture
  }
\end{verbatim}

We add the potential horizontal lines specified by the option \texttt{hlines}.

\begin{verbatim}
\bool_if:NT \l_@@_hlines_bool
{
\end{verbatim}

44
The counter \c@iRow has the value \(-1\) only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

\begin{verbatim}
\int_compare:nNnT \c@iRow > { -1 }
{ \bool_if:NF \g_@@_row_of_col_done_bool
  { \int_compare:nNnF \c@iRow = \l_@@_last_row_int
    { \bool_if:NTF \c_@@_colortbl_loaded_bool
      { \CT@arc@ \hrule height \arrayrulewidth } } \hrule height \arrayrulewidth
    } }
}
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_array:
{ \box_clear_new:N \l_@@_cell_box
  \cs_if_exist:NT \theiRow
  { \int_set_eq:NN \l_@@_save_iRow_int \c@iRow }
  \int_gzero_new:N \c@iRow
  \cs_if_exist:NT \thejCol
  { \int_set_eq:NN \l_@@_save_jCol_int \c@jCol }
  \int_gzero_new:N \c@jCol
  \normalbaselines

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

  \arraystretch

  \arraycolsep

  \\tabskip = \c_zero_skip
\end{verbatim}

If the option small is used, we have to do some tuning. In particular, we change the value of \arraystretch (this parameter is used in the construction of \@arstrutbox in the beginning of \{array\}).

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

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

\begin{verbatim}
\cs_set:Npn \ialign
{ \bool_if:NTF \c_@@_colortbl_loaded_bool
  { \CT@everycr
    { \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: } \@@_everycr:
      \\everycr: }
    \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
\@@_everycr:
}
  \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
\@@_everycr:
}
\end{verbatim}

The box \@arstrutbox is a box constructed in the beginning of the environment \{array\}. The construction of that box takes into account the current values of \arraystretch and \extrarowheight of \{array\}. That box is inserted (via \@arstrut) in the beginning of each row of the array. That’s why we use the dimensions of that box to initialize the variables which will be the dimensions of the box.

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

\footnote{The option small of nicematrix changes (among other) the value of \arraystretch. This is done, of course, before the call of \{array\}.}
potential first and last row of the environment. This initialization must be done after the creation of \arstrutbox and that’s why we do it in the \ialign.

\dim_gzero_new:N \g_@@_dp_row_zero_dim
\dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
\dim_gzero_new:N \g_@@_ht_row_zero_dim
\dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
\dim_gzero_new:N \g_@@_ht_row_one_dim
\dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
\dim_gzero_new:N \g_@@_dp_ante_last_row_dim
\dim_gset:Nn \g_@@_dp_ante_last_row_dim { \box_dp:N \@arstrutbox }
\dim_gzero_new:N \g_@@_ht_last_row_dim
\dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
\dim_gzero_new:N \g_@@_dp_last_row_dim
\dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

After its first use, the definition of \ialign will revert automatically to its default definition. With this programmation, we will have, in the cells of the array, a clean version of \ialign.

\cs_set_eq:NN \ialign \@@_standard_ialign:
\halign}

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

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

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

\cs_set_eq:NN \@@_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 \firsthline \hline
\cs_set_eq:NN \lasthline \hline
\cs_set_eq:NN \Ldots \@@_Ldots
\cs_set_eq:NN \Cdots \@@_Cdots
\cs_set_eq:NN \Vdots \@@_Vdots
\cs_set_eq:NN \Ddots \@@_Ddots
\cs_set_eq:NN \Iddots \@@_Iddots
\cs_set_eq:NN \hdottedline \@@_hdottedline:
\cs_set_eq:NN \Hspace \@@_Hspace:
\cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
\cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
\cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
\cs_set_eq:NN \onlyMainNiceMatrix \@@_onlyMainNiceMatrix:n
\cs_set_eq:NN \CodeAfter \@@_CodeAfter:n
\bool_if:NT \l_@@_renew_dots_bool
\begin{enumerate}
\item \cs_set_eq:NN \ldots \@@_Ldots
\item \cs_set_eq:NN \cdots \@@_Cdots
\item \cs_set_eq:NN \vdots \@@_Vdots
\item \cs_set_eq:NN \\ldots \@@_ldots
\item \cs_set_eq:NN \\ldots \@@_ldots
\item \cs_set_eq:NN \\cdots \@@_cdots
\item \cs_set_eq:NN \\vdots \@@_Vdots
\item \cs_set_eq:NN \\ldots \@@_ldots
\end{enumerate}

The sequence \g_@@_multicolumn_cells_seq will contain the list of the cells of the array where a command \multicolumn\{n\}{...}{...} with n > 1 is issued. In \g_@@_multicolumn_sizes_seq,

33The user will probably not employ directly \ialign in the array... but more likely environments that utilize \ialign internally (e.g.: \{substack\}).
the “sizes” (that is to say the values of $n$) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

The counter $\c@iRow$ will be used to count the rows of the array (its incrementation will be in the first cell of the row).

At the end of the environment \{array\}, $\c@iRow$ will be the total number de rows. \g@@rowtotalint will be the number or rows excepted the last row (if $\l@@last_row_bool$ has been raised with the option last-row).

The counter $\c@jCol$ will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter $\g@@col_total_int$. These counters are updated in the command \@@Cell: executed at the beginning of each cell.

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

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

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

\begin{verbatim}
\l_@@_letter_for_dotted_lines_str \l_@@_letter_for_dotted_lines_str \exp_args:NV \newcolumntype \l_@@_letter_for_dotted_lines_str {
!\}
\int_compare:nNnF \c@iRow = 0 { \skip_horizontal:N 2\l_@@_radius_dim }
\int_compare:nNnF \c@iRow = \l_@@_last_row_int { }

Consider the following code:

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

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

\begin{verbatim}
\int_compare:nNnT \c@jCol > \g_@@_last_vdotted_col_int { \int_gset_eq:NN \g_@@_last_vdotted_col_int \c@jCol \tl_gput_right:Nx \g_@@_internal_code_after_tl { \@@_vdottedline:n { \int_use:N \c@jCol } } \int_gzero_new:N \g_@@_last_vdotted_col_int \bool_if:NT \c_@@_siunitx_loaded_bool \@@_renew_NC@rewrite@S: \bool_gset_false:N \g_@@_last_col_found_bool \end{verbatim}

During the construction of the array, the instructions \texttt{\Cdots}, \texttt{\Ldots}, etc. will be written in token lists \texttt{\g_@@_Cdots_lines_tl}, etc. which will be executed after the construction of the array.

\begin{verbatim}
\tl_gclear_new:N \g_@@_Cdots_lines_tl \tl_gclear_new:N \g_@@_Ldots_lines_tl \tl_gclear_new:N \g_@@_Vdots_lines_tl \tl_gclear_new:N \g_@@_Ddots_lines_tl \tl_gclear_new:N \g_@@_Iddots_lines_tl \tl_gclear_new:N \g_@@_Hdotsfor_lines_tl
\end{verbatim}
The environment \texttt{\textbackslash NewDocumentEnvironment \{ \textbackslash NiceArrayWithDelims \} \{ m\ m \ O \} \ m \ O \} \}

\setlength{\l_@@_left_delim_tl}{#1}
\setlength{\l_@@_right_delim_tl}{#2}
\setboolean\g_@@_row_of_col_done\false
\ifstr\empty\g_@@_name_env\g_@@_name_env
\set\g_@@_name_env{NiceArrayWithDelims}
\adapt\S\column:
\ifmath\false\fataln{Yet\text{-}in\text{-}env}
\set\l_@@_in_env\true
We deactivate Tikz externalization because we will use PGF pictures with the options \texttt{overlay} and \texttt{remember picture} (or equivalent forms).
\if\tikzlibraryexternal\false\else\fi
\ifstandalone\false\else\fi
We increment the counter \texttt{\g_@@_env_int} which counts the environments of the package.
\incr\g_@@_env_int
\if\l_@@_block_auto_columns\false\else\fi
We do a redefinition of \texttt{\@arrayrule} because we want that the vertical rules drawn by | in the preamble of the array don’t extend in the potential exterior rows.
\set\g_@@_ht_last_row{\max\boxht\l_@@_cell_box}
\set\g_@@_dp_last_row{\max\boxdp\l_@@_cell_box}
A value of \texttt{-1} for the counter \texttt{\l_@@_last_row_int} means that the user has used the option \texttt{last-row} without value, that is to say without specifying the number of that last row. In this case, we try to read that value from the aux file (if it has been written on a previous run).
\compare\l_@@_last_row_int\leq{-2}
\else
\put\l_@@_update_for_first_and_last_row:
\set\g_@@_ht_last_row{\max\boxht\l_@@_cell_box}
\set\g_@@_dp_last_row{\max\boxdp\l_@@_cell_box}
\endif
A value based on the name is more reliable than a value based on the number of the environment.
\compare\l_@@_last_row_int\geq{-1}
\else
\set\l_@@_last_row_int\use{\g_@@_name_env}
\endif
A value of \texttt{-1} for the counter \texttt{\l_@@_last_row_int} means that the user has used the option \texttt{last-row} without value, that is to say without specifying the number of that last row. In this case, we try to read that value from the aux file (if it has been written on a previous run).
\compare\l_@@_last_row_int\leq{-2}
\else
\set\l_@@_update_for_first_and_last_row:
\set\g_@@_ht_last_row{\max\boxht\l_@@_cell_box}
\set\g_@@_dp_last_row{\max\boxdp\l_@@_cell_box}
\endif
A value based on the name is more reliable than a value based on the number of the environment.
\compare\l_@@_last_row_int\geq{-1}
\else
\set\l_@@_last_row_int\use{\g_@@_name_env}
\endif
A value of $-1$ for the counter $\l_@@_last_col_int$ means that the user has used the option last-col without value, that is to say without specifying the number of that last column. In this case, we try to read that value from the aux file (if it has been written on a previous run).

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

We compute the width of the two delimiters.

The command \bBigg@ is a command of amsmath.

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

We construct the preamble of the array in $\l_\tmpa_tl$. 

\begin{verbatim}
\cs_if_exist:cT { @@_last_row_ \l_@@_name_str }
{
  \int_set:Nn \l_@@_last_row_int
  \begin{use:c { @@_last_row_ \l_@@_name_str } }

\end{use:c { @@_last_row_ \l_@@_name_str } }
\}

\cs_if_exist:cT { @@_last_col_ \l_@@_name_str }
{
  \int_set:Nn \l_@@_last_col_int
  \begin{use:c { @@_last_col_ \l_@@_name_str } }

\end{use:c { @@_last_col_ \l_@@_name_str } }
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
Here is the beginning of the box which will contain the array. The \hbox_set_end: corresponding to this \hbox_set:Nw will be in the second part of the environment (and the closing \c_math_toggle_token also).

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

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

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

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

\int_compare:nNnT \l_@@_last_row_int > { -2 }
{ 
  \bool_if:NF \l_@@_last_row_without_value_bool 
  { \int_compare:nNnF \l_@@_last_row_int = \c@iRow 
    { \@@_error:n { Wrong-last-row } 
      \int_gset_eq:NN \l_@@_last_row_int \c@iRow 
    } 
  } 
}
Now, the definition of \texttt{\@jCol} and \texttt{\@@col_total_int} change: \texttt{\@jCol} will be the number of
columns without the “last column”; \texttt{\@@col_total_int} will be the number of columns with this
“last column”:\footnote{We remind that the potential “first column” (exterior) has the number 0.}
\begin{verbatim}
\int_gset_eq:NN \@jCol \@@col_total_int
\bool_if:nT \@@last_col_found_bool { \int_gdecr:N \@jCol }
\end{verbatim}
We fix also the value of \texttt{\@iRow} and \texttt{\@@row_total_int} with the same principle.
\begin{verbatim}
\int_gset_eq:NN \@@row_total_int \@iRow
\int_compare:nNnT \l@@last_row_int > {-1} { \int_gdecr:N \@iRow }
\end{verbatim}

Now, we begin the real construction in the output flow of TeX. First, we take into account
a potential “first column” (we remind that this “first column” has been constructed in an overlapping
position and that we have computed its width in \texttt{\@@width_first_col_dim}: see p. \pageref{page59}).
\begin{verbatim}
\int_compare:nNnT \l@@first_col_int = 0
{ \skip_horizontal:N \arraycolsep
 \skip_horizontal:N \@@width_first_col_dim }
\end{verbatim}
The construction of the real box is different in \texttt{NiceArray} and in the other environments because,
in \texttt{NiceArray}, we have to take into account the value of \texttt{baseline} and we have no delimiter to
put. We begin with \texttt{NiceArray}:
\begin{verbatim}
\bool_if:NTF \l@@NiceArray_bool
{ \pgfpicture \@@_qpoint: { row - 1 }
 \dim_gset_eq:NN \@tmpa_dim \pgf@y
 \@@_qpoint: { row - \int_use:N \@iRow - base }
 \dim_gsub:Nn \@tmpa_dim \pgf@y
 \endpgfpicture
 \int_compare:nNnT \l@@first_row_int = 0
{ \skip_horizontal:N \@@ht_row_zero_dim \@@dp_row_zero_dim }
\box_move_up:nn \@tmpa_dim \box_use_drop:N \l@@the_array_box }
\end{verbatim}
We convert a value of \texttt{t} to a value of 1.
\begin{verbatim}
\str_if_eq:VnTF \l@@baseline_str { b }
{ \pgfpicture \@@_qpoint: { row - 1 }
 \dim_gset_eq:NN \@tmpa_dim \pgf@y
 \@@_qpoint: { row - \int_use:N \@iRow - base }
 \dim_gsub:Nn \@tmpa_dim \pgf@y
 \endpgfpicture
 \int_compare:nNnT \l@@first_row_int = 0
{ \skip_horizontal:N \@@ht_row_zero_dim \@@dp_row_zero_dim }
\box_move_up:nn \@tmpa_dim \box_use_drop:N \l@@the_array_box }
\end{verbatim}
We convert the value of \texttt{\l@@baseline_str} (which should represent an integer) to an integer
stored in \texttt{\l@@baseline_int}.
\begin{verbatim}
\int_set:Nn \l@@baseline_int \l@@baseline_str
\bool_if:nT
{ \int_compare_p:nNn \l@@baseline_int < \l@@first_row_int
 | | \int_compare_p:nNn \l@@baseline_int > \l@@row_total_int }
{ \@@_error:n { bad-value-for-baseline }
 \int_set:Nn \l@@baseline_int 1 }
\end{verbatim}
Now, in the case of an environment {pNiceArray}, {bNiceArray}, etc. We compute \l_tmpa_dim which is the total height of the “first row” above the array (when the key first-row is used).

\int_compare:nNnTF \l_@@_first_row_int = 0
{ \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
\box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_@@_the_array_box }
\dim_gset_eq:NN \g_tmpa_dim \pgf@y
\@@_qpoint: { row - 1 }
\dim_gsub:Nn \g_tmpa_dim \pgf@y
\@@_qpoint: { row - \int_use:N \l_tmpa_int - base }
\endpgfpicture
\int_compare:nNnT \l_@@_first_row_int = 0
{ \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim + \g_@@_dp_row_zero_dim }
\box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_@@_the_array_box }
\dim_gset_eq:NN \g_tmpa_dim \pgf@y
\@@_qpoint: { row - 1 }
\dim_gsub:Nn \g_tmpa_dim \pgf@y
\@@_qpoint: { row - \int_use:N \l_tmpa_int - base }
\endpgfpicture

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

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

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

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

\skip_vertical:N -\l_tmpa_dim
\hbox
{ \skip_horizontal:N -\arraycolsep
\box_use_drop:N \l_@@_the_array_box
\skip_horizontal:N -\arraycolsep
}
\skip_vertical:N -\l_tmpb_dim
\hbox
{ \c_math_toggle_token
\left #1
\vcenter
\right #2
\c_math_toggle_token
}

A value of −1 for \l_@@_last_row_int means that there is a “last row” but the user have not set the value with the option last row (and we are in the first compilation).
We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in \g_@@_width_last_col_dim: see p. 60).

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

The command \@@_put_box_in_flow: puts the box \l_tmpa_box (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the depth to take back into account the potential exterior rows (the total height of the first row has been computed in \l_tmpa_dim and the total height of the potential last row in \l_tmpb_dim).

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

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

We take into account the position of the mathematical axis.

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

We will compute the real width of both delimiters used.

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

The construction of the array in the environment {NiceArrayWithDelims} is, in fact, done by the environment {@@-light-syntax} or by the environment {@@-normal-syntax} (whether the option light-syntax is used or not). When the key light-syntax is not used, the construction is a standard environment (and, thus, it’s possible to use verbatim in the array).

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

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

\exp_args:NV \@@_array: \l_tmpa_tl
\@@_create_col_nodes:
endarray

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

\NewDocumentEnvironment { @@-light-syntax } { b }
{
First, we test whether the environment is empty. It’s only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in #1.

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

The body of the environment, which is stored in the argument #1, is now splitted into items (and not tokens)
\seq_gclear_new:N \g_@@_rows_seq
\tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
\exp_args:NVN \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

If the environment uses the option last-row without value (i.e. without saying the number of the rows), we have now the opportunity to know that value. We do it, and so, if the token list \l_@@_code_for_last_row_tl is not empty, we will use directly where it should be.
\int_compare:nNnT \l_@@_last_row_int = { -1 }
{ \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }

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

\exp_args:NV \@@_array: \l_tmpa_tl
We need a global affection because, when executing \l_tmpa_tl, we will exit the first cell of the array.
\seq_gclear_new:N \g_@@_rows_seq \l_tmpa_tl
\exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
\seq_gclear_new:N \g_@@_cells_seq
\seq_gset_split:Nnn \g_@@_cells_seq \l_@@_end_of_row_tl { - } { #1 }
\seq_gpop_left:NN \g_@@_cells_seq \l_@@_end_of_row_tl
\seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
\@@_create_col_nodes:
endarray

Now, the second part of the environment. It is empty. That’s not surprising because we have caught the whole body of the environment with the specifier b provided by xparse.
}
\cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
{\tl_if_empty:nF { #1 }
\seq_gclear_new:N \g_@@_cells_seq
\seq_gset_split:Nnn \g_@@_cells_seq \l_@@_end_of_row_tl { - } { #1 }
\seq_gpop_left:NN \g_@@_cells_seq \l_@@_end_of_row_tl
{\seq_map_function:NN \g_@@_cells_seq \@@_cell_with_light_syntax:n
\@@_create_col_nodes:
endarray

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

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

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

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

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

```
\cs_new:Npn \@@_create_col_nodes:
{ \crcr \int_compare:nNnT \c@iRow = 0 { \@@_fatal:n { Zero~row } } \int_compare:nNnT \l_@@_first_col_int = 0 { \omit \skip_horizontal:N -2\arraycolsep \pgfpicture \pgfrememberpicturepositiononpagetrue \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin \str_if_empty:NF \l_@@_name_str { \pgfnodealias { \@@_env: - col - 0 } { \l_@@_name_str - col - 0 } } \endpgfpicture } \omit \pgfpicture \pgfrememberpicturepositiononpagetrue \pgfcoordinate { \@@_env: - col - 1 } \pgfpointorigin \str_if_empty:NF \l_@@_name_str { \pgfnodealias { \@@_env: - col - 1 } { \l_@@_name_str - col - 1 } } \endpgfpicture & }
```

The following instruction must be put after the instruction \omit. \bool_gset_true:N \g_@@_row_of_col_done_bool

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

```
\pgfpicture \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin \str_if_empty:NF \l_@@_name_str { \pgfnodealias { \@@_env: - col - 0 } { \l_@@_name_str - col - 0 } } \endpgfpicture
```

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

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

```
\skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
```

```
\bool_if:NF \l_@@_auto_columns_width_bool { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
{ \bool_lazy_and:nnTF \l_@@_auto_columns_width_bool \l_@@_block_auto_columns_width_bool { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim } { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
```
We begin a loop over the columns. The integer \texttt{\g_tmpa_int} will be the number of the current column. This integer is used for the Tikz nodes.

\begin{verbatim}
\int_gset:Nn \g_tmpa_int 1
\bool_if:NTF \g_@@_last_col_found_bool
  { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
  { \prg_replicate:nn { \g_@@_col_total_int - 1 } }
{ }
\omit
\int_gincr:N \g_tmpa_int
\skip_horizontal:N \g_tmpa_skip
\hbox
{ \pgfpicture
  \pgfrememberpicturepositiononpagetrue
  \pgfcoordinate { \@@_env: - col - 2 } \pgfpointorigin
  \str_if_empty:NF \l_@@_name_str
  { \pgfnodealias { \@@_env: - col - 2 } \l_@@_name_str - col - 2 }
  \endpgfpicture }
\endverbatim

The incrementation of the counter \texttt{\g_tmpa_int} must be done after the \texttt{\omit} of the cell.

\begin{verbatim}
\int_gincr:N \g_tmpa_int
\skip_horizontal:N \g_tmpa_skip
\hbox
{ \pgfpicture
  \pgfrememberpicturepositiononpagetrue
  \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
  \pgfpointorigin
  \str_if_empty:NF \l_@@_name_str
  { \pgfnodealias { \@@_env: - col - \@@_succ:n \g_tmpa_int }
    { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
  }
  \endpgfpicture }
\endverbatim

\begin{verbatim}
\bool_if:NT \g_@@_last_col_found_bool
  { \skip_horizontal:N 2\arraycolsep
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
    \pgfpointorigin
    \str_if_empty:NF \l_@@_name_str
    { \pgfnodealias { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
      { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
    }
    \endpgfpicture
  \skip_horizontal:N -2\arraycolsep
  }
\endverbatim

Here is the preamble for the “first column” (if the user uses the key \texttt{first-col})

\begin{verbatim}
\tl_const:Nn \c_@@_preamble_first_col_tl
{ > }
\end{verbatim}
\@_begin_of_row:
The contents of the cell is constructed in the box \l_@@_cell_box because we have to compute some dimensions of this box.
\hbox_set:Nw \l_@@_cell_box \c_math_toggle_token \bool_if:NT \l_@@_small_bool \scriptstyle
We insert \l_@@_code_for_first_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”:
\bool_lazy_and:nnT \{ \int_compare_p:nNn \c@iRow > 0 \} \bool_lazy_or_p:nn \{ \int_compare_p:nNn \l_@@_last_row_int < 0 \} \{ \int_compare_p:nNn \c@iRow < \l_@@_last_row_int \} \{ \l_@@_code_for_first_col_tl \} \xglobal \colorlet { nicematrix-first-col } { . }
Be careful: despite this letter l the cells of the “first column” are composed in a R manner since they are composed in a \hbox_overlap_left:n.
\tl_const:Nn \c_@@_preamble_last_col_tl \{ \int_compare_p:nNn \c@jCol > 1 \} \{ \c_math_toggle_token \hbox_set_end: \@@_update_for_first_and_last_row:
We actualise the width of the “first column” because we will use this width after the construction of the array.
\dim_gset:Nn \g_@@_width_first_col_dim \{ \dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box } \}
The content of the cell is inserted in an overlapping position.
\hbox_overlap_left:n \{ \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim \@@_node_for_the_cell: \{ \box_use_drop:N \l_@@_cell_box \} \skip_horizontal:N \l_@@_left_delim_dim \skip_horizontal:N \l_@@_left_margin_dim \skip_horizontal:N \l_@@_extra_left_margin_dim \} \skip_horizontal:N -2\arraycolsep
Here is the preamble for the “last column” (if the user uses the key last-col).
\tl_const:Nn \c_@@_preamble_last_col_tl \{ \int_compare_p:nNn \c@jCol > 0 \}
With the flag \g_@@_last_col_found_bool, we will know that the “last column” is really used.
\bool_gset_true:N \g_@@_last_col_found_bool \int_gincr:N \c@jCol \int_gset_eq:NN \g_@@_col_total_int \c@jCol
The contents of the cell is constructed in the box \l_tmpa_box because we have to compute some dimensions of this box.
\hbox_set:Nw \l_@@_cell_box \c_math_toggle_token \bool_if:NT \l_@@_small_bool \scriptstyle
We insert \_@@\_code\_for\_last\_col\_tl... but we don’t insert it in the potential “first row” and in the potential “last row”:

\begin{verbatim}
\int_compare:nNnT \c@iRow > 0
\{
  \bool_lazy_or:nnT
  \{
    \int_compare_p:nNn \l_@@_last_row_int < 0
  \}
  \{
    \l_@@_code_for_last_col_tl
    \xglobal \colorlet { nicematrix-last-col } { . }
  \}
\}
\end{verbatim}

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

\begin{verbatim}
\dim_gset:Nn \g_@@_width_last_col_dim { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
\end{verbatim}

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

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

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

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

We put . and . for the delimiters but, in fact, that doesn’t matter because these arguments won’t be used in \{NiceArrayWithDelims\} (because the flag \_@@\_NiceArray_bool is raised).

\begin{verbatim}
\NiceArrayWithDelims . .
\end{verbatim}

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

\begin{verbatim}
\NewDocumentEnvironment { pNiceArray } { }
\{
  \str_if_empty:NT \g_@@_name_env_str
  \{
    \str_gset:Nn \g_@@_name_env_str { pNiceArray }
  \}
  \@@_test_if_math_mode:
  \NiceArrayWithDelims ( )
\end{verbatim}
The environment \{NiceMatrix\} and its variants

\cs_new_protected:Npn \@@_define_env:n #1
\NewDocumentEnvironment { \#1 NiceMatrix } { ! O { } }
\str_gset:Nn \g_@@_name_env_str \#1 NiceMatrix
\tl_set:Nn \l_@@_type_of_col_tl C
\keys_set:nn { NiceMatrix / NiceMatrix } { \#1 }
\exp_args:Nnx \@@_begin_of_NiceMatrix:nn { \#1 } \l_@@_type_of_col_tl
\end { \#1 NiceArray } 
\end { \#1 NiceArrayWithDelims }
After the construction of the array

\cs_new_protected:Npn \@@_after_array:
{
\group_begin:

When the option \texttt{last-col} is used in the environments with explicit preambles (like \{\texttt{NiceArray}\}, \{\texttt{pNiceArray}\}, etc.) a special type of column is used at the end of the preamble in order to compose the cells in an overlapping position (with \texttt{\hbox\_overlap\_right:n}) but (if \texttt{last-col} has been used), we don’t have the number of that last column. However, we have to know that number for the color of the potential \texttt{\textbackslash Vdots} drawn in that last column. That’s why we fix the correct value of \texttt{\l_@@\_last\_col\_int} in that case.

\bool_if:NT \g_@@\_last\_col\_found_bool
{ \int_set_eq:NN \l_@@\_last\_col\_int \g_@@\_col\_total\_int }

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

\bool_if:NT \l_@@\_last\_col\_without\_value_bool
{ \dim_set_eq:NN \l_@@\_last\_col\_int \g_@@\_col\_total\_int

\iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { \_@@\_last\_col\_ \int_use:N \g_@@\_env\_int }
  { \int_use:N \g_@@\_col\_total\_int }
}
\str_if_empty:NF \l_@@\_name_str
{ \iow_shipout:Nx \@mainaux
  { \cs_gset:cpn { \_@@\_last\_col\_ \l_@@\_name\_str }
    { \int_use:N \g_@@\_col\_total\_int }
  }
}
\iow_shipout:Nn \@mainaux \ExplSyntaxOff
}

It’s also time to give to \texttt{\l_@@\_last\_row\_int} its real value. But, if the user had used the option \texttt{last-row} without value, we write in the aux file the number of that last row for the next run.

\bool_if:NT \l_@@\_last\_row\_without\_value_bool
{ \dim_set_eq:NN \l_@@\_last\_row\_int \g_@@\_row\_total\_int

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

\bool_if:NT \l_@@\_light\_syntax\_bool
{ \iow_shipout:Nn \@mainaux \ExplSyntaxOn
\iow_shipout:Nx \@mainaux
{ \cs_gset:cpn { \@\_last\_row\_ \int_use:N \g_@@\_env\_int }
  { \int_use:N \g_@@\_row\_total\_int }
}
\str_if_empty:NF \l_@@\_name_str
{ \iow_shipout:Nx \@mainaux
  { \cs_gset:cpn { \_@@\_last\_row\_ \l_@@\_name\_str }
    { \int_use:N \g_@@\_row\_total\_int }
  }
}

If the environment has a name, we also write a value based on the name because it’s more reliable than a value based on the number of the environment.
By default, the diagonal lines will be parallelized. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

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

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

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

Now, we really draw the lines.

We draw the vertical rules of the option `vlines` before the `internal-code-after` because the option `white` of a `\Block` may have to erase these vertical rules.
We recall that, when externalization is used, \texttt{\tikzpicture} and \texttt{\endtikzpicture} (or \texttt{\pgfpicture} and \texttt{\endpgfpicture}) must be directly “visible”. That’s why we have to define the adequate version of \texttt{\@@\_draw\_dotted\_lines}: whether Tikz is loaded or not (in that case, only PGF is loaded).

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

\[
\begin{pmatrix}
a + b + c & a + b & a \\
a & a + b & a + b + c
\end{pmatrix}
\]

The command \texttt{\@@\_find\_extremities\_of\_line:nnnn} takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
• the third argument is the \( x \)-value of the orientation vector of the line;
• the fourth argument is the \( y \)-value of the orientation vector of the line.

This command computes:

• \( \texttt{l@@}\_initial\_i\_int \) and \( \texttt{l@@}\_initial\_j\_int \) which are the coordinates of one extremity of the line;
• \( \texttt{l@@}\_final\_i\_int \) and \( \texttt{l@@}\_final\_j\_int \) which are the coordinates of the other extremity of the line;
• \( \texttt{l@@}\_initial\_open\_bool \) and \( \texttt{l@@}\_final\_open\_bool \) to indicate whether the extremities are open or not.

\begin{verbatim}
1475 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
1476 { 
First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.
1477 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { } 

Initialization of variables.
1478 \int_set:Nn \l_@@_initial_i_int { #1 }
1479 \int_set:Nn \l_@@_initial_j_int { #2 }
1480 \int_set:Nn \l_@@_final_i_int { #1 }
1481 \int_set:Nn \l_@@_final_j_int { #2 }

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean \( \texttt{l@@}\_stop\_loop\_bool \) will be used to control these loops. In the first loop, we search the “final” extremity of the line.
1482 \bool_set_false:N \l_@@_stop\_loop\_bool
1483 \bool_do_until:Nn \l_@@_stop\_loop\_bool
1484 { 
1485 \int_add:Nn \l_@@_final_i_int { #3 }
1486 \int_add:Nn \l_@@_final_j_int { #4 }

We test if we are still in the matrix.
1487 \bool_set_false:N \l_@@_final\_open\_bool
1488 \int_compare:nNnTF \l_@@_final\_i\_int > \c@iRow
1489 { 
1490 \int_compare:nNnTF \l_@@_final\_i\_int > \c@iRow
1491 { \bool_set_true:N \l_@@_final\_open\_bool }
1492 { \int_compare:nNnTF \l_@@_final\_j\_int > \c@jCol
1493 { \bool_set_true:N \l_@@_final\_open\_bool }
1494 }
1495 }
1496 { 
1497 \int_compare:nNnTF \l_@@_final\_j\_int < 1
1498 { \int_compare:nNnT \l_@@_final\_j\_int > \c@jCol
1499 { \bool_set_true:N \l_@@_final\_open\_bool }
1500 }
1501 }
1502 { 
1503 \int_compare:nNnTF \l_@@_final\_j\_int = 1
1504 { \int_compare:nNnT \l_@@_final\_j\_int = 1
1505 { \bool_set_true:N \l_@@_final\_open\_bool }
1506 }
1507 }
1508 \bool_if:TF \l_@@_final\_open\_bool

If we are outside the matrix, we have found the extremity of the dotted line and it’s an open extremity.
1509 { 

65
\end{verbatim}
We do a step backwards.

\bool_set_true:N \l_@@_stop_loop_bool

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

\cs_if_exist:cTF
  \bool_set_true:N \l_@@_stop_loop_bool
  \bool_set_true:N \l_@@_final_open_bool
\cs_if_exist:cTF
  \bool_set_true:N \l_@@_stop_loop_bool

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be mark as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environnement), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

\cs_set:cpn
  \l_@@_dotted_
  \l_@@_final_i_int - \l_@@_final_j_int
\cs_set:cpn
  \l_@@_dotted_
  \l_@@_initial_i_int - \l_@@_initial_j_int

For \l_@@_initial_i_int and \l_@@_initial_j_int the programmation is similar to the previous one.
The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

We remind that, when there is a “last row” \l_@@_last_row_int will always be (after the construction of the array) the number of that “last row” even if the option last-row has been used without value.

The command \@@_actually_draw_Ldots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.
The following function is also used by \dotsfor.
\cs_new_protected:Npn \@@_actually_draw_Ldots:\{
\bool_if:NTF \l_@@_initial_open_bool
{\%}
\bool_if:NTF \l_@@_final_open_bool
{\%}
\dim_add:Nn \l_@@_x_initial_dim \arraycolsep
\dim_add:Nn \l_@@_y_initial_dim \pgf\ydim
}\dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
\@@_draw_line:
\W e raise the line of a quantity equal to the radius of the dots because we want the dots really “on”
the line of texte.
\dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
\@@_draw_line:
\}

The first and the second arguments are the coordinates of the cell where the command has been
issued. The third argument is the list of the options.
\cs_new_protected:Npn \@@_draw_Cdots:nnn \#1 \#2 \#3
{\%}
\\keys_set:nn { NiceMatrix / xdots } { \#3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\}

The previous command may have changed the current environment by marking some cells as “dotted”,
but, fortunately, it is outside the group for the options of the line.
\group_begin:
\int_compare:nNnTF { \#1 } = 0
{\color { nicematrix-first-row } }
{\%}
\group_end:

We remind that, when there is a “last row” \l_@@_last_row_int will always be (after the construction
of the array) the number of that “last row” even if the option last-row has been used without value.
\int_compare:nNnT { \#1 } = \l_@@_last_row_int
{\color { nicematrix-last-row } }
\}
\keys_set:nn { NiceMatrix / xdots } { \#3 }
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:

The command \@@_actually_draw_Cdots: has the following implicit arguments:
• \l_@@_initial_i_int
• \l_@@_initial_j_int
• \l_@@_initial_open_bool
\texttt{\textbackslash \_\_\_final\_i\_int}
\texttt{\textbackslash \_\_\_final\_j\_int}
\texttt{\textbackslash \_\_\_final\_open\_bool.}

\begin{Verbatim}
\cs_new_protected:Npn \@\_actually_draw_Cdots:
\begin{verbatim}
\begin{Verbatim}
1703 \cs_new_protected:Npn \@\_actually_draw_Cdots:
1704 { \bool_if:NTF \l_@@_initial_open_bool
1705 { \@@_qpoint: \{ col - \int_use:N \l_@@_initial_j_int \}
1706 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
1707 \dim_add:Nn \l_@@_x_initial_dim \arraycolsep
1708 } \{ \@@_set_initial_coords_from_anchor:n \{ mid-east \} \}
1709 \bool_if:NTF \l_@@_final_open_bool
1710 { \@@_qpoint: \{ col - \@@_succ:n \l_@@_final_j_int \}
1711 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
1712 \dim_sub:Nn \l_@@_x_final_dim \arraycolsep
1713 } \{ \@@_set_final_coords_from_anchor:n \{ mid-west \} \}
1714 \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
1715 { \@@_qpoint: \{ row - \int_use:N \l_@@_initial_i_int \}
1716 \dim_set_eq:NN \l_tmpa_dim \pgf@y
1717 \@@_qpoint: \{ row - \@@_succ:n \l_@@_initial_i_int \}
1718 \dim_set:Nn \l_@@_y_initial_dim { \l_tmpa_dim + \pgf@y \} / 2 \}
1719 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
1720 \@@_actually_draw_Cdots:
1721 \bool_if:NTF \l_@@_initial_open_bool \l_@@_final_open_bool
1722 { \@@_qpoint: \{ row - \int_use:N \l_@@_final_i_int \}
1723 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
1724 \bool_if:NTF \l_@@_final_open_bool
1725 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim \}
1726 \@@_draw_line:
1727 \end{verbatim}
\end{Verbatim}
\end{verbatim}

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

\begin{Verbatim}
\cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
\begin{verbatim}
\begin{Verbatim}
1737 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
1738 { \tl_if_empty:VF \l_@@_xdots_color_tl \{ \color { \l_@@_xdots_color_tl \}
1739 \cs_if_free:cT \{ \_dotted \} \{ \_ \} #1 \#2 \}
1740 \keys_set:nn { NiceMatrix / xdots } { #3 }
1741 \@@_find_extremities_of_line:nnnn \{ #1 \} \{ #2 \} 1 0
\end{verbatim}
\end{verbatim}

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\begin{Verbatim}
\begin{Verbatim}
1744 \int_compare:nNnTF \{ #2 \} = 0
1745 { \color { nicematrix-first-col \}
1746 \int_compare:nNnT \{ #2 \} = \l_@@_last_col_int
1747 { \color { nicematrix-last-col \}
1748 \keys_set:nn \{ NiceMatrix / xdots \} \{ #3 \}
1749 \@@_actually_draw_Vdots:
1750 \end{verbatim}
\end{verbatim}

\end{Verbatim}
The command `\ @@_actually_draw_Vdots: ` has the following implicit arguments:

- `\ @@_initial_i_int`
- `\ @@_initial_j_int`
- `\ @@_initial_open_bool`
- `\ @@_final_i_int`
- `\ @@_final_j_int`
- `\ @@_final_open_bool`

```
cs_new_protected:Npn \ @@_actually_draw_Vdots:
  {
    The boolean \l_tmpa_bool indicates whether the column is of type `l` (of \{NiceArray\}) or may be considered as if.
    \bool_set_false:N \l_tmpa_bool
    \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
    \{
      \@@_set_initial_coords_from_anchor:n { south-west }
      \@@_set_final_coords_from_anchor:n { north-west }
      \bool_set:Nn \l_tmpa_bool { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
    }
    Now, we try to determine whether the column is of type `c` (of \{NiceArray\}) or may be considered as if.
    \bool_if:NTF \l_@@_initial_open_bool
    \{
      \@@_qpoint: { row - 1 }
      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \}
    \bool_if:NTF \l_@@_final_open_bool
    \{
      \@@_set_initial_coords_from_anchor:n { south }
    \}
    \bool_if:NTF \l_@@_final_open_bool
    \{
      \@@_qpoint: { col - \int_use:N \l_@@_initial_j_int }
      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      \ @@_qpoint: { col - \@@_succ:n \l_@@_initial_j_int }
      \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \{
      \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
    }
    \}
    \bool_if:NTF \l_@@_final_open_bool
    \{
      \@@_set_initial_coords_from_anchor:n { north }
    \}
    \bool_if:NTF \l_@@_initial_open_bool
    \{
      \@@_qpoint: { col - \int_use:N \l_@@_initial_j_int }
      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      \@@_qpoint: { col - \@@_succ:n \l_@@_initial_j_int }
      \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \{
      \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
    }
    \}
    \bool_if:NTF \l_@@_final_open_bool
    \{
      \@@_set_final_coords_from_anchor:n { north } \l_@@_y_final_dim \pgf@y
    \}
    \{
      \@@_set_final_coords_from_anchor:n { north }
    \}
    \bool_if:NTF \l_@@_initial_open_bool
    \{
      \@@_qpoint: { col - \int_use:N \l_@@_initial_j_int }
      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
      \@@_qpoint: { col - \@@_succ:n \l_@@_initial_j_int }
      \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \{
      \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
    }
    \}
    \bool_if:NTF \l_@@_final_open_bool
    \{
      \@@_set_final_coords_from_anchor:n { north } \l_@@_y_final_dim \pgf@y
    \}
    \{
      \@@_set_final_coords_from_anchor:n { north }
    \}
    \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
    \{
      \dim_set:Nn \l_@@_x_initial_dim
      \{
        \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
    ```
For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

\begin{Verbatim}
\begin{Verbatim}
\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
\{ \cs_if_free:cT { @@_dotted_#1 - #2 } \{ \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 1
\end{Verbatim}
\end{Verbatim}
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

The command \texttt{\hdots} has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

\begin{Verbatim}
\begin{Verbatim}
\cs_new_protected:Npn \@@_actually_draw_Ddots: \{ \bool_if:NTF \l_@@_initial_open_bool \{ \@@_qpoint: { row - \int_use:N \l_@@_initial_i_int } \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y \@@_qpoint: { col - \int_use:N \l_@@_initial_j_int } \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x \} \{ \@@_set_initial_coords_from_anchor:n { south-east } \} \bool_if:NTF \l_@@_final_open_bool \{ \@@_qpoint: { row - \@@_succ:n \l_@@_final_i_int } \dim_set_eq:NN \l_@@_y_final_dim \pgf@y \@@_qpoint: { col - \@@_succ:n \l_@@_final_j_int } \dim_set_eq:NN \l_@@_x_final_dim \pgf@x \} \{ \@@_set_final_coords_from_anchor:n { north-west } \} \end{Verbatim}
\end{Verbatim}
We have retrieved the coordinates in the usual way (they are stored in \l_@@_x_initial_dim, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

\bool_if:NT \l_@@_parallelize_diags_bool
\{\
\int_gincr:N \g_@@_ddots_int\
\}

We test if the diagonal line is the first one (the counter \g_@@_ddots_int is created for this usage).
\int_compare:nNnTF \g_@@_ddots_int = 1
\{\
\dim_gset:Nn \g_@@_delta_x_one_dim { \l_@@_x_final_dim - \l_@@_x_initial_dim }\
\dim_gset:Nn \g_@@_delta_y_one_dim { \l_@@_y_final_dim - \l_@@_y_initial_dim }\
\}

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate \l_@@_x_initial_dim.
\{\
\dim_set:Nn \l_@@_y_final_dim { \l_@@_y_initial_dim + ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) * \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim }\
\}
\@@_draw_line:
\}

We draw the \iddots diagonals in the same way. The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
\{\
\cs_if_free:cT { @@ _ dotted _ #1 - #2 }
\{\
\@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }\
\}
\}
\group_begin:
\keys_set:nn { NiceMatrix / xdots } { #3 }\
\tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }\
\@@_actually_draw_Iddots:
\group_end:
\}

The command \@@_actually_draw_Iddots: has the following implicit arguments:
- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int

73
The command \texttt{\textbackslash NiceMatrixLastEnv} is not used by the package \texttt{nicematrix}. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name).

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

The actual instructions for drawing the dotted line with Tikz

The command \texttt{\_\_\_\_\_draw_line:} should be used in a \texttt{\{pgfpicture\}}. It has six implicit arguments:

- \texttt{\_\_\_\_\_x\_initial\_dim}
- \texttt{\_\_\_\_\_y\_initial\_dim}
- \texttt{\_\_\_\_\_x\_final\_dim}
- \texttt{\_\_\_\_\_y\_final\_dim}
- \texttt{\_\_\_\_\_initial\_open\_bool}
We have to do a special construction with \exp_args:NV to be able to put in the list of options in the correct place in the Tikz instruction.

We have used the fact that, in PGF, a color name can be put directly in a list of options (that’s why we have put directly \l_@@_xdots_color_tl).

The argument of \@@_draw_non_standard_dotted_line:n is, in fact, the list of options.

The command \@@_draw_standard_dotted_line: draws the line with our system of points (which give a dotted line with real round points).

First, we put the labels.
The dimension $\l_@@_l_dim$ is the length $\ell$ 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{\left((\l_@@_x_final_dim - \l_@@_x_initial_dim \right)^2 + \left((\l_@@_y_final_dim - \l_@@_y_initial_dim \right)^2}}}}

It seems that, during the first compilations, the value of $\l_@@_l_dim$ may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.
The integer $\l_tmpa_int$ is the number of dots of the dotted line.

```latex
\bool_if:NTF \l_@@_initial_open_bool
\bool_if:NTF \l_@@_final_open_bool
\int_set:Nn \l_tmpa_int
{ \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
\int_set:Nn \l_tmpa_int
{ \dim_ratio:nn
  \l_@@_l_dim - \l_@@_xdots_shorten_dim
  \l_@@_inter_dots_dim }
\bool_if:NTF \l_@@_final_open_bool
\int_set:Nn \l_tmpa_int
{ \dim_ratio:nn
  \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim
  \l_@@_inter_dots_dim }
```

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

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

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

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

In the loop over the dots, the dimensions $\l_@@_x_initial_dim$ and $\l_@@_y_initial_dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```latex
\dim_gadd:Nn \l_@@_x_initial_dim
```
User commands available in the new environments

The commands \@@_Ldots, \@@_Cdots, \@@_Vdots, \@@_Ddots and \@@_Iddots will be linked to \Ldots, \Cdots, \Vdots, \Ddots and \Iddots in the environments \{NiceArray\} (the other environments of nicematrix rely upon \{NiceArray\}). The starred versions of these commands are deprecated since version 3.1 but, as for now, they are still available with an error.

The syntax of these commands uses the character \_ as embellishment and that’s why we have to insert a character \_ in the arg spec of these commands. However, we don’t know the future catcode of \_ in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates \_). That’s why these commands will be defined in a \AtBeginDocument and the arg spec will be rescanned.

\AtBeginDocument

\exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl

\exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl

\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl

\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_cdots }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
{\bool_if:nTF { #1 }
{ \@@_error:n { starred~commands } }
{ \@@_instruction_of_type:nn { Vdots } { #2 , down = #3 , up = #4 } }
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_vdots }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
{\bool_if:nTF { #1 }
{ \@@_error:n { starred~commands } }
{ \@@_instruction_of_type:nn { Ddots } { #2 , down = #3 , up = #4 } }
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_ddots }
\bool_gset_true:N \g_@@_empty_cell_bool
}

\exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
{\bool_if:nTF { #1 }
{ \@@_error:n { starred~commands } }
{ \@@_instruction_of_type:nn { Iddots } { #2 , down = #3 , up = #4 } }
\bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_iddots }
\bool_gset_true:N \g_@@_empty_cell_bool
}

End of the \AtBeginDocument.

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

\cs_new_protected:Npn \@@_Hspace:
{ \bool_gset_true:N \g_@@_empty_cell_bool
\hspace
}

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

\cs_set_eq:NN \@@_old_multicolumn \multicolumn
\cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
{ \@@_old_multicolumn { #1 } { #2 } { #3 }
\int_compare:nNnT #1 > 1
{ \seq_gput_left:Nx \g_@@_multicolumn_cells_seq \int_eval:n { \c@iRow - \int_use:N \c@jCol } \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
\int_gadd:N \c@jCol { #1 - 1 }
}
}

The command \@@_Hdotsfor will be linked to \Hdotsfor in \{NiceArrayWithDelims\}. This command uses an optional argument (as does \hdotsfor) but this argument is discarded (in \hdotsfor, this argument is used for fine tuning of the space between two consecutive dots). Tikz nodes are created also the implicit cells of the \Hdotsfor (maybe we should modify that point).
This command must not be protected since it begins with \multicolumn.

\cs_new:Npn \@@_Hdotsfor:
\{ \multicolumn { 1 } { C } { } \}
\@@_Hdotsfor_i

\ExplSyntaxOff
\def \@tempa { O { } m O { } E { _ ^ } { { } { } } }
\ExplSyntaxOn
\tl_set_eq:NN \l_@@_a_signature_tl \@tempa
\ExplSyntaxOff
\def \l_@@_argspec_tl { O { } m O { } E { _ ^ } { { } { } } }
\tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
\bool_if:NTF \c_@@_draft_bool
\{ We don’t put ! before the last optional argument for homogeneity with \Cdots, etc. which have only one optional argument.
\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \tl_gput_right:Nx \g_@@_Hdotsfor_lines_tl
\{ \@@_Hdotsfor:nnnn \int_use:N \c@iRow \int_use:N \c@jCol \#2 \#1, \#3, down = \#4, up = \#5 \}
\prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\}
\AtBeginDocument
\\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \tl_gput_right:Nx \g_@@_Hdotsfor_lines_tl
\{ \@@_Hdotsfor:nnnn \int_use:N \c@iRow \int_use:N \c@jCol \#2 \#1, \#3, down = \#4, up = \#5 \}
\prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\}
\}
\AtBeginDocument
\\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\\exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
\{ \tl_gput_right:Nx \g_@@_Hdotsfor_lines_tl
\{ \@@_Hdotsfor:nnnn \int_use:N \c@iRow \int_use:N \c@jCol \#2 \#1, \#3, down = \#4, up = \#5 \}
\prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { C } { } } \}
\}
\}
\}
\end{verbatim}

End of \AtBeginDocument.

\cs_new_protected:Npn \@@_Hdotsfor:nnnn \#1 \#2 \#3 \#4
\{ \bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
\}
\bool_set:Nn \l_@@_initial_i_int { \#1 }
\bool_set_eq:Nn \l_@@_final_i_int \l_@@_initial_i_int

For the row, it’s easy.
\int_set:Nn \l_@@_initial_i_int { \#1 }
\int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

For the column, it’s a bit more complicated.
\int_compare:nNnTF \#2 = 1
\{ \int_set:Nn \l_@@_initial_j_int { \#1 }
\bool_set_true:N \l_@@_initial_open_bool
\} \cs_if_exist:cTF
\pgf @ sh @ ns @ \@@_env:
\int_use:N \l_@@_initial_i_int

80
We declare all the cells concerned by the \Hdotsfor as “dotted” (for the dotted lines created by \Cdots, \Ldots, etc., this job is done by \@@_find_extremities_of_line:nnnn). This declaration is done by defining a special control sequence (to nil).

\int_step_inline:nnn { #2 } { #2 + #3 - 1 } { \cs_set:cpn { @@_dotted #1 - ##1 } { } }

The control sequence \@@_rotate: will be linked to \rotate in {NiceArrayWithDelims}. The command will exit three levels of groups in order to execute the command “\box_rotate:Nn \l_@@_cell_box { 90 }” just after the construction of the box \l_@@_cell_box.

\cs_new_protected:Npn \@@_rotate: { \group_insert_after:N \@@_rotate_i: }
\cs_new_protected:Npn \@@_rotate_i: { \group_insert_after:N \@@_rotate_ii: }
\cs_new_protected:Npn \@@_rotate_ii: { \group_insert_after:N \@@_rotate_iii: }
\cs_new_protected:Npn \@@_rotate_iii: { \box_rotate:Nn \l_@@_cell_box { 90 } }

If we are in the last row, we want all the boxes composed with the command \rotate aligned upwards.

\int_compare:nNnT \c@iRow = \l_@@_last_row_int
{ \vbox_set_top:Nn \l_@@_cell_box
{ \vbox_to_zero:n \{} }
0.8 \text{ ex} \text{ will be the distance between the principal part of the array and our element (which is composed with } \texttt{rotate}.

\begin{verbatim}
\skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 \text{ ex} }
\box_use:N \l_@@_cell_box
\}
\}
\}
\end{verbatim}

The command \texttt{\line} accessible in code-after

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

First, we write a command with an argument of the format \texttt{i-j} and applies the command \texttt{\int_eval:n} to \texttt{i} and \texttt{j}; this must \textit{not} be protected (and is, of course fully expandable).

\begin{verbatim}
\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
\{ \int_eval:n { #1 } - \int_eval:n { #2 } \}
\ExplSyntaxOff
\def \@tempa { O { } m m ! O { } E { _ ^ } { { } { } } }
\ExplSyntaxOn
\tl_set_eq:NN \l_@@_a_signature_tl \@tempa
\end{verbatim}

With the following construction, the command \texttt{@@_double_int_eval:n} is applied to both arguments before the application of \texttt{@@_line_i:nn} (the construction uses the fact the \texttt{@@_line_i:nn} is protected and that \texttt{@@_double_int_eval:n} is fully expandable).

\begin{verbatim}
\AtBeginDocument
\{ \tl_set:Nn \l_@@_argspec_tl { O { } m m ! O { } E { _ ^ } { { } { } } } \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
\{ \group_begin: \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 } \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } } \use:x
\{ \@@_line_i:nn #2 \q_stop \}
\{ \@@_double_int_eval:n #3 \q_stop \}
\group_end: \}
\bool_if:NTF \c_@@_draft_bool
\{ \cs_new_protected:Npn \@@_line_i:nn #1 #2 { } \}
\{ \cs_new_protected:Npn \@@_line_i:nn #1 #2
\{ \bool_set_false:N \l_@@_initial_open_bool \bool_set_false:N \l_@@_final_open_bool \bool_if:nTF
\{ \cs_if_free_p:c { pgf @ sh @ ns \@@_env: - #1 } \|
\{ \cs_if_free_p:c { pgf @ sh @ ns \@@_env: - #2 } \}
\end{verbatim}

\textsuperscript{37} Indeed, we want that the user may use the command \texttt{\line} in code-after with LaTeX counters in the arguments — with the command \texttt{\value}.
We recall that, when externalization is used, \tikzpicture and \endtikzpicture (or \pgfpicture and \endpgfpicture) must be directly “visible” and that why we do this static construction of the command \@draw_line_ii::

\c_@@_pgfortikzpicture_tl
\@draw_line_iii:nn { #1 } { #2 }
\c_@@_endpgfortikzpicture_tl

The following command must be protected (it’s used in the construction of \@draw_line_ii:nn).

\cs_new_protected:Npn \@draw_line_iii:nn #1 #2

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

The vertical rules

We give to the user the possibility to define new types of columns (with \newcolumntype of \array) for special vertical rules (e.g. rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command \OnlyMainNiceMatrix in that goal. However, that command must be no-op outside the environments of nicematrix (and so the user will be allowed to use the same new type of column in the environments of nicematrix and in the standard environments of \array).

That’s why we provide first a global definition of \OnlyMainNiceMatrix.

\cs_set_eq:NN \OnlyMainNiceMatrix \use:n

Another definition of \OnlyMainNiceMatrix will be linked to the command in the environments of nicematrix. Here is that definition, called \@@_OnlyMainNiceMatrix:n.

\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1

This definition may seem complicated by we must remind that the number of row \c@iRow is incremented in the first cell of the row, after an potential vertical rule on the left side of the first cell.

The command \@@_OnlyMainNiceMatrix_i:n is only a short-cut which is used twice in the above command. This command must not be protected.

\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
\{\int_compare:nNnF \c@iRow = 0 \{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int \{ #1 \} \} \}

Remember that \c@iRow is not always inferior to \l_@@_last_row_int because \l_@@_last_row_int may be equal to −2 or −1 (we can’t write \int_compare:nNnT \c@iRow < \l_@@_last_row_int).

In fact, independently of \OnlyMainNiceMatrix, which is a convenience given to the user, we have to modify the behaviour of the standard specifier “|”. Remark first that the natural way to do that would be to redefine the specifier “|” with \newcolumntype:

\newcolumntype { | } { ! { \OnlyMainNiceMatrix \vline } }

However, this code fails if the user uses \DefineShortVerb{|} of fancyvrb. Moreover, it would not be able to deal correctly with two consecutive specifiers “|” (in a preamble like ccc||ccc).

That’s why we have done a redefinition of the macro \@arrayrule of \array and this redefinition will add \@@_vline: instead of \vline to the preamble (that definition is in the beginning of \{NiceArrayWithDelims\}).

Here is the definition of \@@_vline:. This definition must be protected because you don’t want that macro expanded during the construction of the preamble (the tests in \@@_OnlyMainNiceMatrix:n must be effective in each row and not once for all when the preamble is constructed).

\cs_new_protected:Npn \@@_vline: { \@@_OnlyMainNiceMatrix:n { \@@_vline_i: } }

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

\cs_set_eq:NN \@@_vline_i: \vline

The command \@@_draw_vlines will be executed when the user uses the option vlines (which draws all the vlines of the array).

\cs_new_protected:Npn \@@_draw_vlines:
\{\group_begin:

The command \CT@arc@ is a command of color from \colortbl.

\bool_if:NT \c_@@_colortbl_loaded_bool \CT@arc@
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\pgfsetlinewidth \arrayrulewidth
First, we compute in \l_tmpa_dim the height of the rules we have to draw.

\@@_qpoint: {row - 1 }
\dim_set_eq:NN \l_tmpa_dim \pgf@y
\pgfusepathqfill
\@@_qpoint: { row - \@@_succ:n \c@iRow }
\dim_sub:Nn \l_tmpa_dim \pgf@y
\pgfusepathqfill
\pgfsetlinewidth \arrayrulewidth
We translate vertically to take into account the potential “last row”.

\dim_zero:N \l_tmbp_dim
\int_compare:nNnT \l_@@_last_row_int > { -1 } \{ \dim_set_eq:NN \l_tmbp_dim \g_@@_dp_last_row_int \dim_add:Nn \l_tmbp_dim \g_@@_ht_last_row_dim \}

84
We adjust the value of $\text{l\_tmpa\_dim}$ by the width of the horizontal rule just before the “last row”.

\begin{verbatim}
\@@_qpoint: { row - \@@_succ:n\c@iRow }
\dim_add:Nn \l\_tmpa\_dim \pgfy
\@@_qpoint: { row - \@@_succ:n \g\_\_row\_total\_int }
\dim_sub:Nn \l\_tmpa\_dim \pgfy
\dim_sub:Nn \l\_tmpa\_dim \l\_tmpb\_dim
\end{verbatim}

Now, we can draw the lines with a loop.

\begin{verbatim}
\int_step_inline:nnn
\{ \bool_if:NTF \l\_\_0\_NiceArray\_bool 1 2 \}
\{ \bool_if:NTF \l\_\_0\_NiceArray\_bool \{ \@@_succ:n \c@jCol \} \c@jCol \}
\{ \pgfpathmoveto
\\pgfpointadd
\{ \@@_qpoint: { col - ##1 } \}
\{ \pgfpoint
\{ -0.5 \arrayrulewidth
\int_compare:nNnT \{ ##1 \} = 1
\{ \int_compare:nNnT \l\_\_0\_first\_col\_int = 1
\{ + \arrayrulewidth \}
\}
\}
\l\_tmpb\_dim
\}
\pgfpathlineto
\pgfpointadd
\{ \@@_qpoint: { col - ##1 } \}
\{ \pgfpoint
\{ -0.5 \arrayrulewidth
\int_compare:nNnT \{ ##1 \} = 1
\{ \int_compare:nNnT \l\_\_0\_first\_col\_int = 1
\{ + \arrayrulewidth \}
\}
\}
\l\_tmpb\_dim + \l\_tmpa\_dim
\}
\pgfusepathqstroke
\endpgfpicture
\end{verbatim}

The commands to draw dotted lines to separate columns and rows

These commands don’t use the normal nodes, the medium nor the large nodes. They only use the \texttt{col}-nodes and the \texttt{row}-nodes.

\textbf{Horizontal dotted lines}

The following command must \textit{not} be protected because it’s meant to be expanded in a \texttt{\noalign}.

\begin{verbatim}
\bool_if:NTF \c\_\_0\_draft\_bool
\end{verbatim}

85
On the other side, the following command should be protected.

\cs_new_protected:Npn \@@_hdottedline_i:
{
  \noalign { \skip_vertical:N 2\l_@@_radius_dim }
  \@@_hdottedline_i:
}

We write in the code-after the instruction that will eventually draw the dotted line. It’s not possible to draw this dotted line now because we don’t know the length of the line (we don’t even know the number of columns).

\tl_gput_right:Nx \g_@@_internal_code_after_tl
{ \@@_hdottedline:n { \int_use:N \c@iRow } }

The command \@@_hdottedline:n is the command written in the code-after that will actually draw the dotted line. Its argument is the number of the row before which we will draw the row.

\AtBeginDocument
{
  \c_@@_pgfortikzpicture_tl
  \@@_hdottedline_i:n { #1 }
  \c_@@_endpgfortikzpicture_tl
}

The following command must be protected since it is used in the construction of \@@_hdottedline:n.

\cs_new_protected:Npx \@@_hdottedline:n #1
{
  \bool_set_true:N \exp_not:N \l_@@_initial_open_bool
  \bool_set_true:N \exp_not:N \l_@@_final_open_bool
  \c_@@_pgfortikzpicture_tl
  \@@_hdottedline_i:n { #1 }
  \c_@@_endpgfortikzpicture_tl
}

We do a translation par \l_@@_radius_dim because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

\dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
\dim_sub:Nn \l_@@_y_initial_dim \l_@@_radius_dim
\dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim

The dotted line will be extended if the user uses margin (or left-margin and right-margin). The aim is that, by standard the dotted line fits between square brackets (\hline doesn’t).

\begin{bNiceMatrix}
1 & 2 & 3 & 4
\hline
1 & 2 & 3 & 4
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

But, if the user uses margin, the dotted line extends to have the same width as a \hline.
For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \ \l_@@\inter\dots\dim$ is *ad hoc* for a better result.

As for now, we have no option to control the style of the lines drawn by \hdottedline and the specifier \":\" in the preamble. That’s why we impose the style standard.

We recall that, when externalization is used, \tikzpicture and \endtikzpicture (or \pgfpicture and \endpgfpicture) must be directly “visible”.

The command \CT@arc@ is a command of color from colortbl.
We do a translation $-\l_@@_radius_dim$ because we want the dotted line to have exactly the same position as a vertical rule drawn by “|” (considering the rule having a width equal to the diameter of the dots).

$$\dim_set:Nn \l_@@_x_initial_dim { \pgf@x - \l_@@_radius_dim }$$

$$\dim_set:Nn \l_@@_x_final_dim { \pgf@x - \l_@@_radius_dim }$$

$$\ @@_qpoint: { row - 1 }$$

We arbitrary decrease the height of the dotted line by a quantity equal to $\l_@@_inter_dots_dim$ in order to improve the visual impact.

$$\dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_inter_dots_dim }$$

$$\ @@_qpoint: { row - \@@_succ:n \c@iRow }$$

$$\dim_set:Nn \l_@@_y_final_dim { \pgf@y + 0.5 \l_@@_inter_dots_dim }$$

As for now, we have no option to control the style of the lines drawn by \hdottedline and the specifier “:” in the preamble. That’s why we impose the style standard.

$$\tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl$$

$$\@@_draw_line:$$

The environment \{NiceMatrixBlock\}

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

$$\bool_new:N \l_@@_block_auto_columns_width_bool$$

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

$$\keys_define:nn \{ \text{NiceMatrix} / \text{NiceMatrixBlock} \}$$

$$\{ \text{auto-columns-width .code:n =}$$

$$\{ \bool_set_true:N \l_@@_block_auto_columns_width_bool$$

$$\dim_gzero_new:N \g_@@_max_cell_width_dim$$

$$\bool_set_true:N \l_@@_auto_columns_width_bool$$

$$\}$$

$$\}$$

$$\NewDocumentEnvironment \{ \text{NiceMatrixBlock} \} \{ ! O \{ \} \}$$

$$\{ \int_gincr:N \g_@@_NiceMatrixBlock_int$$

$$\dim_zero:N \l_@@_columns_width_dim$$

$$\keys_set:nn \{ \text{NiceMatrix} / \text{NiceMatrixBlock} \} \{ \#1 \}$$

$$\bool_if:NT \l_@@_block_auto_columns_width_bool$$

$$\{ \cs_if_exist:cT \{ @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int \}$$

$$\exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim$$

$$\{ @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int \}$$

$$\}$$

$$\}$$

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

$$\bool_if:NT \l_@@_block_auto_columns_width_bool$$

$$\{ \iow_shipout:Nn \@mainaux \ExplSyntaxOn$$

$$\iow_shipout:Nx \@mainaux$$

88
For technical reasons, we have to include the width of an eventual rule on the right side of the cells.

\begin{Verbatim}
\cs_gset:cpn \{ \@@_max_cell_width \int_use:N \g_@@_NiceMatrixBlock_int \}
\end{Verbatim}

The extra nodes

First, two variants of the functions \texttt{\dim\_min:nn} and \texttt{\dim\_max:nn}.

\begin{Verbatim}
\cs_generate_variant:Nn \dim_min:nn { v n }
\cs_generate_variant:Nn \dim_max:nn { v n }
\end{Verbatim}

We have three macros of creation of nodes: \texttt{\@@\_create\_medium\_nodes:}, \texttt{\@@\_create\_large\_nodes:} and \texttt{\@@\_create\_medium\_and\_large\_nodes:}.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command \texttt{\@@\_computations\_for\_medium\_nodes:} to do these computations.

The command \texttt{\@@\_computations\_for\_medium\_nodes:} must be used in a \texttt{pgfpicture}.

For each row \( i \), we compute two dimensions \( 1_{\@@\_row\_i\_min\_dim} \) and \( 1_{\@@\_row\_i\_max\_dim} \). The dimension \( 1_{\@@\_row\_i\_min\_dim} \) is the minimal \( y \)-value of all the cells of the row \( i \). The dimension \( 1_{\@@\_row\_i\_max\_dim} \) is the maximal \( y \)-value of all the cells of the row \( i \).

Similarly, for each column \( j \), we compute two dimensions \( 1_{\@@\_column\_j\_min\_dim} \) and \( 1_{\@@\_column\_j\_max\_dim} \). The dimension \( 1_{\@@\_column\_j\_min\_dim} \) is the minimal \( x \)-value of all the cells of the column \( j \). The dimension \( 1_{\@@\_column\_j\_max\_dim} \) is the maximal \( x \)-value of all the cells of the column \( j \).

Since these dimensions will be computed as maximum or minimum, we initialize them to \texttt{\c_max_dim} or \texttt{- \c_max_dim}.

\begin{Verbatim}
\cs_new_protected:Npn \@@\_computations\_for\_medium\_nodes: \{ \int_step_variable:nnNn \l_@@\_first\_row\_int \g_@@\_row\_total\_int \@@\_i: \\
\{ \dim_zero_new:c { l_@@\_row\_i\_min\_dim} \dim_set_eq:cN { l_@@\_row\_i\_min\_dim} \c_max_dim \dim_zero_new:c { l_@@\_row\_i\_max\_dim} \dim_set:cn { l_@@\_row\_i\_max\_dim} { - \c_max_dim } \cal_step_variable:nnNn \l_@@\_first\_col\_int \g_@@\_col\_total\_int \@@\_j: \\
\{ \dim_zero_new:c { l_@@\_column\_j\_min\_dim} \dim_set_eq:cN { l_@@\_column\_j\_min\_dim} \c_max_dim \dim_zero_new:c { l_@@\_column\_j\_max\_dim} \dim_set:cn { l_@@\_column\_j\_max\_dim} { - \c_max_dim } \cal_step_variable:nnNn \l_@@\_first\_row\_int \g_@@\_row\_total\_int \@@\_i: \\
\{ \int_step_variable:nnNn \l_@@\_first\_col\_int \g_@@\_col\_total\_int \@@\_j: \\
\}
\}
\end{Verbatim}

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

\begin{Verbatim}
\int_step_variable:nnNn \l_@@\_first\_row\_int \g_@@\_row\_total\_int \@@\_i: \\
\{ \int_step_variable:nnNn \l_@@\_first\_col\_int \g_@@\_col\_total\_int \@@\_j: \\
\}
\end{Verbatim}
If the cell \((i-j)\) is empty or an implicit cell (that is to say a cell after implicit ampersands \&) we don’t update the dimensions we want to compute.

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

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

Now, we have to deal with empty rows or empty columns since we don’t have created nodes in such rows and columns.

Here is the command \texttt{\@@create_medium_nodes}. When this command is used, the “medium nodes” are created.
Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

\tl_set:Nn \l_@@_suffix_tl { -medium }
\@@_create_nodes:
\endpgfpicture

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

\tl_set:Nn \l_@@_suffix_tl { -large }
\@@_create_nodes:
\endpgfpicture

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

\int_set:Nn \l_@@_first_row_int 1
\int_set:Nn \l_@@_first_col_int 1
\int_step_variable:nNn { \c@iRow - 1 } \@@_i:
\dim_set:cn { \l_@@_row _ \@@_i: _ min _ dim }
\dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } +
\dim_use:c { \l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
\dim_set_eq:cc { \l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }

\textsuperscript{38}If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`.

\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\@@_computations_for_medium_nodes:
\@@_computations_for_large_nodes:
\tl_set:Nn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture
The control sequence \@@_create_nodes: is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions l_@@_row_i: _min_dim, l_@@_row_i: _max_dim, l_@@_column_j: _min_dim and l_@@_column_j: _max_dim. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses \l_@@_suffix_tl (medium or large).

We draw the rectangular node for the cell (\@@_i-\@@_j).

Now, we create the nodes for the cells of the \texttt{multicolumn}. We recall that we have stored in \g_@@_multicolumn_cells_seq the list of the cells where a \texttt{\textbackslash multicolumn}{\ldots}{\ldots} with n>1 was issued and in \g_@@_multicolumn_sizes_seq the correspondent values of n.
The command \@@_node_for_multicolumn:nn takes two arguments. The first is the position of the cell where the command \multicolumn{n}{...}{...} was issued in the format \textit{i-j} and the second is the value of \textit{n} (the length of the “multi-cell”).

\begin{verbatim}
\cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
{ \@@_extract_coords_values: #1 \q_stop
  \@@_i: { #1 }
  \@@_j: { #2 }
}
\end{verbatim}

Block matrices

The code in this section is for the construction of block matrices. It has no direct link with the environment \texttt{NiceMatrixBlock}. The following command will be linked to \texttt{Block} in the environments of \texttt{nicematrix}. We define it with \texttt{NewDocumentCommand} of \texttt{xparse} because it has an optional argument between < and > (for TeX instructions put before the math mode of the label).

\begin{verbatim}
\NewDocumentCommand \@@_Block: { O { } m D < > { } m }
{ \@@_Block_i #2 \q_stop { #1 } { #3 } { #4 } }
\end{verbatim}

The first mandatory argument of \@@_Block: has a special syntax. It must be of the form \textit{i-j} where \textit{i} and \textit{j} are the size (in rows and columns) of the block.

\begin{verbatim}
\cs_new:Npn \@@_Block_i #1-#2 \q_stop
{ \@@_Block_ii:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { \str_if_empty:NF \l_@@_name_str }
{ \pgfnodealias \\
  \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
{ \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
}
\end{verbatim}

Now, the arguments have been extracted: \#1 is \textit{i} (the number of rows of the block), \#2 is \textit{j} (the number of columns of the block), \#3 is the list of key-values, \#4 are the tokens to put before the math mode and \#5 is the label of the block.

\begin{verbatim}
\tl_gput_left:Nx \g_@@_internal_code_after_tl
\tl_gput_left:Nx \g_@@_internal_code_after_tl
\end{verbatim}

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

\begin{verbatim}
\tl_gput_left:Nx \g_@@_internal_code_after_tl
\end{verbatim}
It’s not allowed to use the command \Block twice in the same cell of the array. That’s why, at the first use, we link the command \Block to a special version. The scope of this link is the cell of the array:

```latex
\cs_set_eq:NN \Block \@@_Block_error:nn
\cs_new:Npn \@@_Block_error:nn #1 #2
{\@@_error:n { Second~Block }
\cs_set_eq:NN \Block \use:nn}
```

The following command \@@_Block_iii:nnnnnn will be used in the `code-after`:

```latex
\cs_new_protected:Npn \@@_Block_iii:nnnnnn #1 #2 #3 #4 #5 #6
{\group_begin:
\keys_set:nn { NiceMatrix / Block } { #5 }
\bool_if:nTF
{ \int_compare_p:nNn { #3 } > \c@iRow
|| \int_compare_p:nNn { #4 } > \c@jCol }
{ \msg_error:nnnn { nicematrix } { Block~too~large } { #1 } { #2 } }
\group_end:}
```

We put the contents of the cell in the box \l_@@_cell_box because we want the command \rotate used in the content to be able to rotate the box.

```latex
\hbox_set:Nn \l_@@_cell_box { #6 }
```

The construction of the node corresponding to the merged cells.

```latex
\pgfpicture
\pgfresetboundingbox
\pgf@relevantforpicturesizefalse
\@@_qpoint: { row - #1 }
\dim_set_eq:NN \l_tmpa_dim \pgf@y
\@@_qpoint: { col - #2 }
\dim_set_eq:NN \l_tmpb_dim \pgf@x
\@@_qpoint: { row - \@@_succ:n { #3 } }
\dim_set_eq:NN \l_tmpc_dim \pgf@y
\@@_qpoint: { col - \@@_succ:n { #4 } }
\dim_set_eq:NN \l_tmpd_dim \pgf@x
```

The following code doesn’t work for the first vertical rule. You should allow the option `white` if and only if the option `vlines` and `hlines` has been used.

```latex
\bool_if:NT \l_@@_white_bool
{ \begin { pgfscope }
\pgfsetfillcolor { white }
```
Usually, the vertical rules are before the col-nodes. But there is an exception: if there is no “first col”, the first vertical rule is after the col node.\footnote{That’s true for the vertical rules drawn by “|” due to the conception of \{array\} (of array) and we have managed to have the same behaviour with \texttt{vlines}.}

Since we don’t want the white rectangle to erase a part of this first rule, we have to do an adjustment in this case, after the “col node”.

\begin{verbatim}
\int_compare:NnT \{ #2 \} = 1
\{
\int_compare:NnT \l_@@_first_col_int = 1
\{
\dim_add:Nn \l_tmpb_dim \arrayrulewidth \\
\pgfpathrectanglecorners
\{ \pgfpoint \l_tmpb_dim \l_tmpa_dim - \arrayrulewidth \}
\{ \pgfpoint \l_tmpd_dim - \arrayrulewidth \l_tmpc_dim \}
\pgfusepathqfill \\
\end{pgfscope}
\}
\end{verbatim}

We construct the node for the block with the name (#1-#2-block).

The function \texttt{\@@_pgf_rect_node:nnnnn} takes as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

\begin{verbatim}
\begin{pgfscope}
\exp_args:Nx \pgfset{\l_@@_tikz_tl}
\@@_pgf_rect_node:nnnnn
\{ \@@_env: - #1 - #2 - block \\
\l_tmpb_dim \l_tmpa_dim \l_tmpd_dim \l_tmpc_dim \\
\end{pgfscope}
\end{verbatim}

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function \texttt{\@@_pgf_rect_node:nnnnn} takes as arguments the name of the node and two \texttt{pgf} points.

\begin{verbatim}
\bool_if:NT \l_@@_medium_nodes_bool
\{
\@@_pgf_rect_node:nnn
\{ \@@_env: - #1 - #2 - block - medium \\
\pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } \\
\pgfpointanchor { \@@_env: - #3 - #4 - medium } { south-east } \\
\}
\}
\end{verbatim}

Now, we will put the label of the block.

\begin{verbatim}
\int_compare:NnTF \{ #1 \} = \{ #3 \}
\{
\pgfextracty \l_tmpa_dim { \@@_qpoint: { row - #1 - base } }
\pgftransformshift { \pgfpointanchor \l_tmpa_dim { \@@_oqpoint: { row - #1 - base } } }
\pgfnode { rectangle } { base }
{ \box_use_drop:N \l_@@_cell_box } { } { }
\}
\end{verbatim}

If the number of rows is different of 1, we put the label of the block in the center of the node (the label of the block has been composed in \texttt{\l_@@_cell_box}).

\begin{verbatim}
\pgftransformshift { \@@_oqpoint: { #1 - #2 - block } }
\pgfnode { rectangle } { center }
{ \box_use_drop:N \l_@@_cell_box } { } { }
\end{verbatim}
How to draw the dotted lines transparently

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

Automatic arrays

\cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
\{ 
\int_set:Nn \l_@@_nb_rows_int { #1 }
\int_set:Nn \l_@@_nb_cols_int { #2 }
\}
\NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } } 
\{ 
\int_zero_new:N \l_@@_nb_rows_int
\int_zero_new:N \l_@@_nb_cols_int
\@@_set_size:n #4 \q_stop
\begin { NiceArrayWithDelims } { #1 } { #2 }
\{ * { \l_@@_nb_cols_int } { C } \} [ #3 , #5 , #7 ]
\int_compare:nNnT \l_@@_first_row_int = 0 
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 { & }
\prg_replicate:nn \{ \l_@@_nb_cols_int \} \{ \hat \} \}
\int_compare:nNnT \l_@@_last_col_int > { -1 } \{ \hat \} \}
\prg_replicate:nn \l_@@_nb_rows_int 
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ \hat \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ \hat \} \#
\int_compare:nNnT \l_@@_last_col_int > { -1 } \{ \hat \} \}
\}
\int_compare:nNnT \l_@@_last_row_int > { -2 } 
\{ 
\int_compare:nNnT \l_@@_first_col_int = 0 \{ \hat \}
\prg_replicate:nn \{ \l_@@_nb_cols_int - 1 \} \{ \hat \}
\int_compare:nNnT \l_@@_last_col_int > { -1 } \{ \hat \} \}
\}

You put \{ \} before \#6 to avoid a hasty expansion of an eventual \texttt{\arabic{iRow}} at the beginning of the row which would result in an incorrect value of that \texttt{iRow} (since \texttt{iRow} is incremented in the first cell of the row of the \texttt{\halign}).
The command \CodeAfter

The command \CodeAfter catches everything until the end of the current environment (of \nicematrix). First, we go until the next command \end.

We catch the argument of the command \end (in #1).

We process the options

We process the options when the package is loaded (with \usepackage) but we recommend to use \NiceMatrixOptions instead.

We must process these options after the definition of the environment \{NiceMatrix\} because the option renew-matrix executes the code \cs_set_eq:NN \env@matrix \NiceMatrix.

Of course, the command \NiceMatrix must be defined before such an instruction is executed.
Error messages of the package

The following command converts all the elements of a sequence (which are token lists) into strings.
\begin{verbatim}
cs_new_protected:Npn \@@_convert_to_str_seq:N #1 {
\seq_clear:N \l_tmpa_seq
\seq_map_inline:Nn #1 {
 \seq_put_left:Nx \l_tmpa_seq { \tl_to_str:n { ##1 } }
}
\seq_set_eq:NN #1 \l_tmpa_seq
}
\ProcessKeysOptions { NiceMatrix / Package }
\end{verbatim}

The following command creates a sequence of strings (str) from a clist.
\begin{verbatim}
cs_new_protected:Npn \@@_set_seq_of_str_from_clist:Nn #1 #2 {
 \seq_set_from_clist:Nn #1 { #2 }
 \@@_convert_to_str_seq:N #1
}
\@@_set_seq_of_str_from_clist:Nn \c_@@_types_of_matrix_seq {
 NiceMatrix , pNiceMatrix , bNiceMatrix , vNiceMatrix , BNiceMatrix , VNiceMatrix
}
\end{verbatim}

If the user uses too much columns, the command \@@_error_too_much_cols: is executed. This command raises an error but try to give the best information to the user in the error message. The command \seq_if_in:NVTF is not expandable and that’s why we can’t put it in the error message itself. We have to do the test before the \@@_fatal:n.
\begin{verbatim}
cs_new_protected:Npn \@@_error_too_much_cols: {
 \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str {
 \int_compare:nNnTF \l_@@_last_col_int = { -2 } {
 \@@_fatal:n { too-much-cols-for-matrix } }
 \bool_if:NF \l_@@_last_col_without_value_bool {
 \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
 }
 \@@_fatal:n { too-much-cols-for-array }
}
\end{verbatim}

The following command must not be protected since it’s used in an error message.
\begin{verbatim}
cs_new:Npn \@@_message_hdotsfor: {
 \tl_if_empty:VF \g_@@_Hdotsfor_lines_tl {
 -Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
}
\end{verbatim}
\@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
{ You-try-to-use-more-columns-than-allowed-by-your-
 \@@_full_name_env:/@@_message_hdotsfor:\ The-maximal-number-of-
columns-is-\int_eval:n { \l_@@_last_col_int - 1 }-(plus-the-potential-
exterior-ones).-This-error-is-fatal. }
\@@_msg_new:nn { too-much-cols-for-matrix }
{ You-try-to-use-more-columns-than-allowed-by-your-
 \@@_full_name_env:/@@_message_hdotsfor:\ Recall-that-the-maximal-
number-of-columns-for-a-matrix-is-fixed-by-the-LaTeX-counter-
'MaxMatrixCols'.-Its-actual-value-is-\int_use:N \c@MaxMatrixCols.-
This-error-is-fatal. }
\@@_msg_new:nn { too-much-cols-for-array }
{ You-try-to-use-more-columns-than-allowed-by-your-
 \@@_full_name_env:/@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
\int_eval:n { \c@jCol - 1 }-(plus-the-potential-exterior-ones).-
This-error-is-fatal. }
\@@_msg_new:nn { bad-option-for-line-style }
{ Since-you-haven't-loaded-Tikz,-the-only-value-you-can-give-to-'line-style'-
is-'standard'.-If-you-go-on,-this-option-will-be-ignored. }
\@@_msg_new:nn { Unknown-option-for-xdots }
{ As-for-now-there-is-only-three-options-available-here:-'color',-'line-style'-
and-'shorten'-(and-you-try-to-use-'\l_keys_key_str').-If-you-go-on,-
this-option-will-be-ignored. }
\@@_msg_new:nn { ampersand-in-light-syntax }
{ You-can't-use-an-ampersand-(\token_to_str &) to-separate-columns-because
-you-have-used-the-option-'light-syntax'.-This-error-is-fatal. }
\@@_msg_new:nn { double-backslash-in-light-syntax }
{ You-can't-use-'\token_to_str:N \l_@@_end_of_row_tl' to-separate-rows-because-you-have-used-
the-option-'light-syntax'.-You-must-use-the-character-'\l_@@_end_of_row_tl'-(set-by-the-option-'end-of-row').-This-error-is-fatal. }
\@@_msg_new:nn { starred-commands }
{ The-starred-versions-of-'\token_to_str:N \Cdots,'-'\token_to_str:N \Ldots,'-'\token_to_str:N \Vdots,'-'\token_to_str:N \Ddots and-'\token_to_str:N \l_@@_end_of_row_tl'
are-deprecated.-However,-you-can-go-on-for-this-time.-If-you-don't-want-to-
see-this-error-we-should-load-'nicematrix' with-the-option-
'starred-commands'. }
\@@_msg_new:nn { bad-value-for-baseline }
{ The-value-given-to-'baseline'-(\int_use:N \l_tampa_int) is-not-
valid.-The-value-must-be-between-\int_use:N \l_@@_first_row_int and-
\int_use:N \g_@@_row_total_int or-equal-to-'t',-'c' or-'b'.}
If you go on, a value of 1 will be used.

\@@_msg_new:nn { Second-Block }
{
  You can't use \token_to_str:N \Block twice in the same cell of the array.\\
  If you go on, this command (and the other) will be ignored.
}

\@@_msg_new:nn { empty-environment }
{ Your \@@_full_name_env:\ is empty. This error is fatal. }

\@@_msg_new:nn { unknown-cell-for-line-in-code-after }
{ Your command \token_to_str:N \line\{#1\}\{#2\} in the 'code-after'-
  can't be executed because a cell doesn't exist.\\
  If you go on, this command will be ignored.
}

\@@_msg_new:nn { last-col-non-empty-for-NiceArray }
{ In the \@@_full_name_env:, you must use the option-
  'last-col' without value.\\
  However, you can go on for this time-
  (the value '"l_keys_value_tl' will be ignored).
}

\@@_msg_new:nn { Block-too-large }
{ You try to draw a block in the cell #1-#2 of your matrix but
  the matrix is too small for that block.\\
  If you go on, this command will be ignored.
}

\@@_msg_new:nn { Wrong-last-row }
{ You have used 'last-row=\int_use:N \l_@@_last_row_int'-but your-
  \@@_full_name_env:\ seems to have \int_use:N \c@iRow rows.-
  If you go on, the value of \int_use:N \c@iRow will be used for-
  last-row. You can avoid this problem by using 'last-row'-
  without value (more compilations might be necessary).
}

\@@_msg_new:nn { Yet-in-env }
{ Environments \{NiceArray\}-or \{NiceMatrix\}, etc.-can't be nested.\\
  This error is fatal.
}

\@@_msg_new:nn { Outside-math-mode }
{ The \@@_full_name_env:\ can be used only in math mode-
  (and not in \token_to_str:N \vcenter).\\
  This error is fatal.
}

\@@_msg_new:nn { Bad-value-for-letter-for-dotted-lines }
{ The value of key '"l_keys_key_str'-must be of length 1.\\
  If you go on, it will be ignored.
}

\@@_msg_new:nnn { Unknown-key-for-NiceMatrixOptions }
{ The key '"l_keys_key_str'-is unknown for the command-
  \token_to_str:N \NiceMatrixOptions.\\
  If you go on, it will be ignored.\\
  For a list of the available keys, type H <return>.
}
The available options are (in alphabetic order):

- allow-duplicate-names,
- code-for-first-col,
- code-for-first-row,
- code-for-last-col,
- code-for-last-row,
- create-extra-nodes,
- create-medium-nodes,
- create-large-nodes,
- end-of-row,
- exterior-arraycolsep,
- hlines,
- hvlines,
- left-margin,
- letter-for-dotted-lines,
- light-syntax,
- nullify-dots,
- parallelize-diags,
- renew-dots,
- renew-matrix,
- right-margin,
- small,
- transparent,
- vlines,
- xdots/color,
- xdots/shorten-and-
- xdots/line-style.

\@@_msg_new:nnn { Unknown option for NiceArray }

{ The option '\tl_use:N\l_keys_key_str' is unknown for the environment \{NiceArray\}. \}
If you go on, it will be ignored. \}
For a list of the available options, type H<return>.

{ The available options are (in alphabetic order):
- b,
- baseline,
- c,
- code-after,
- code-for-first-col,
- code-for-first-row,
- code-for-last-col,
- code-for-last-row,
- columns-width,
- create-extra-nodes,
- create-medium-nodes,
- create-large-nodes,
- end-of-row,
- extra-left-margin,
- extra-right-margin,
- first-col,
- first-row,
- hlines,
- hvlines,
- last-col,
- last-row,
- left-margin,
- light-syntax,
- name,
- nullify-dots,
This error message is used for the set of keys \texttt{NiceMatrix/NiceMatrix} and \texttt{NiceMatrix/pNiceArray}
(but not by \texttt{NiceMatrix/NiceArray} because, for this set of keys, there is also the options \texttt{t, c} and \texttt{b}).
\begin{verbatim}
\@@_msg_new:nnn { Unknown-option-for-NiceMatrix }
{
  The-option-'\l_use:N\l_keys_key_str'-is-unknown-for-the-
  \@@_full_name_env:. \\n  If-you-go-on,-it-will-be-ignored. \\n  For-a-list-of-the-available-options,-type-H<-return>.
}
{ The-available-options-are-(in-alphabetic-order):-
  b,-
  baseline,-
  c,-
  code-after,-
  code-for-first-col,-
  code-for-first-row,-
  code-for-last-col,-
  code-for-last-row,-
  columns-width,-
  create-extra-nodes,-
  create-medium-nodes,-
  create-large-nodes,-
  end-of-row,-
  extra-left-margin,-
  extra-right-margin,-
  first-col,-
  first-row,-
  hlines,-
  hvlines,-
  l-=(L),-
  last-col,-
  last-row,-
  left-margin,-
  light-syntax,-
  name,-
  nullify-dots,-
  parallelize-diags,-
  r-=(R),-
  renew-dots,-
  right-margin,-
  small,-
  t,-
  vlines,-
  xdots/color,-
  xdots/shorten-and-
  xdots/line-style.
}
\@@_msg_new:nnn { Duplicate-name }
{
  The-name-'\l_keys_value_tl'-is-already-used-and-you-shouldn't-use-
  the-same-environment-name-twice.-You-can-go-on,-but,-
\end{verbatim}
maybe, you will have incorrect results especially if you use 'columns-width=auto'. If you don't want to see this message again, use the option 'allow-duplicate-names'.

For a list of the names already used, type H<return>.

The names already defined in this document are:
\seq_use:Nnnn \g_@@_names_seq { , - } { , - } { -and- }.

You can't give the value 'auto' to the option 'columns-width' here. If you go on, the option will be ignored.

There is a problem. Maybe you have used l, c and r instead of L, C and R in the preamble of your environment. This error is fatal.

Obsolete environments

The following environments are loaded only when the package nicematrix has been loaded with the option obsolete-environments. However, they will be completely deleted in a future version.
16 History

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency). Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment \{NiceArray\} with column types L, C and R.
Changes between version 1.2 and 1.3

New environment \{pNiceArrayC\} and its variants.
Correction of a bug in the definition of \{BNiceMatrix\}, \{vNiceMatrix\} and \{VNiceMatrix\} (in fact, it was a typo).
Options are now available locally in \{pNiceMatrix\} and its variants.
The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types \texttt{w} and \texttt{W} can now be used in the environments \{NiceArray\}, \{pNiceArrayC\} and its variants with the same meaning as in the package \texttt{array}.
New option \texttt{columns-width} to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of \texttt{nicematrix} were focused on the continuous dotted lines whereas the version 2.0 of \texttt{nicematrix} provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment \{pNiceArrayRC\}. With this new implementation, there is no restriction on the width of the columns.
The package \texttt{nicematrix} no longer loads \texttt{mathtools} but only \texttt{amsmath}.
Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option \texttt{code-for-first-row} is now available in the command \texttt{\NiceMatrixOptions}.
Following a discussion on \TeX\ StackExchange\footnote{\texttt{cf.} \url{tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}}, Tikz externalization is now deactivated in the environments of the package \texttt{nicematrix}.

Changes between version 2.1 and 2.1.2

Option \texttt{draft}: with this option, the dotted lines are not drawn (quicker).

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like \texttt{\Cdots} issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix.
That means that it’s possible to do the following matrix with only a \texttt{\Cdots} command (and a single \texttt{\Vdots}).

\[
\begin{pmatrix}
C_j \\
\vdots \\
a \\
\vdots \\
0
\end{pmatrix}
L_i
\]

\footnote{Before this version, there was an error when using \texttt{nicematrix} with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by \texttt{nicematrix} because they use the options \texttt{overlay} and \texttt{remember picture}.}
Changes between version 2.1.3 and 2.1.4

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

See [www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end](www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end)

Changes between version 2.1.4 and 2.1.5

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

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hshline` of `arydshln`).
Possibility to draw vertical dotted lines to separate columns with the specifier `\:` in the preamble (similar to the classical specifier `\|` and the specifier `\::` of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier `\:` in the preamble.
Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.
Option `hlines`.
A warning is issued when the `draft` mode is used. In this case, the dotted lines are not drawn.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of `\|`) as `\Hdotsfor` does.
Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.
Error message when the user gives an incorrect value for `last-row`.
A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol `\:\:` (in the preamble of the array) and `\line` in `code-after`).
The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by `\|`) are now compatible with the color fixed by `colortbl`.
Correction of a bug: it was not possible to use the colon `\:` in the preamble of an array when `pdflatex` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.
Changes between version 3.2 and 3.3

The options first-row, last-row, first-col and last-col are now available in the environments \NiceMatrix, \pNiceMatrix, \bNiceMatrix, etc.
The option columns-width=auto doesn’t need any more a second compilation.
The options renew-dots, renew-matrix and transparent are now available as package options (as said in the documentation).
The previous version of nicematrix was incompatible with a recent version of expl3 (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange\footnote{cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize}, optimization of Tikz externalization is disabled in the environments of nicematrix when the class standalone or the package standalone is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the code-after was not executed.

Changes between version 3.5 and 3.6

LaTeX counters iRow and jCol available in the cells of the array.
Addition of \normalbaselines before the construction of the array: in environments like \{align\} of amsmath the value of \baselineskip is changed and if the options first-row and last-row were used in an environment of nicematrix, the position of the delimiters was wrong.
A warning is written in the .log file if an obsolete environment is used.
There is no longer artificial errors Duplicate-name in the environments of amsmath.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: code-for-first-col, code-for-last-col, code-for-first-row and code-for-last-row.
New command \pAutoNiceMatrix and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command \Block when the block has only one row. With this programmation, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with \multicolumn. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands \NiceMatrixLastEnv and \OnlyMainNiceMatrix.
New options create-medium-nodes and create-large-nodes.

Changes between version 3.9 and 3.10

New option light-syntax (and end-of-row).
New option dotted-lines-margin for fine tuning of the dotted lines.
Changes between versions 3.10 and 3.11

Correction of a bug linked to \texttt{first-row} and \texttt{last-row}.

Changes between versions 3.11 and 3.12

Command \texttt{\rotate} in the cells of the array.
Options \texttt{vlines}, \texttt{hlines} and \texttt{helines}.
Option \texttt{baseline} for \texttt{\NiceArray} (not for the other environments).
The name of the Tikz nodes created by the command \texttt{\Block} has changed: when the command has been issued in the cell $i\text{-}j$, the name is $i\text{-}j\text{-}block$ and, if the creation of the “medium nodes” is required, a node $i\text{-}j\text{-}block\text{-}medium$ is created.
If the user try to use more columns than allowed by its environment, an error is raised by nicematrix (instead of a low-level error).
The package must be loaded with the option \texttt{obsolete\text{-}environments} if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command \texttt{\rotate} is improved when used in the “last row”.
The option \texttt{dotted\text{-}lines\text{-}margin} has been renamed in \texttt{xdots/shorten} and the options \texttt{xdots/color} and \texttt{xdots/line\text{-}style} have been added for a complete customization of the dotted lines.
In the environments without preamble (\texttt{\NiceMatrix}, \texttt{\pNiceMatrix}, etc.), it’s possible to use the options \texttt{l (\texttt{=L})} or \texttt{r (\texttt{=R})} to specify the type of the columns.
The starred versions of the commands \texttt{\Cdots}, \texttt{\Ldots}, \texttt{\Vdots}, \texttt{\Ddots} and \texttt{\Iddots} are deprecated since the version 3.1 of nicematrix. Now, one should load nicematrix with the option \texttt{starred\text{-}commands} to avoid an error at the compilation.
The code of nicematrix no longer uses Tikz but only PGF. By default, Tikz is not loaded by nicematrix.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on \texttt{stackoverflow}).
Better error messages when the user uses & or \texttt{\\} when \texttt{light\text{-}syntax} is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by \texttt{\Ldots}, \texttt{\Cdots}, \texttt{\Vdots}, \texttt{\Ddots}, \texttt{\Iddots}, \texttt{\Hdotsfor} and the command \texttt{\line} in the \texttt{code\text{-}after} with the tokens _ and ^.
The option \texttt{baseline} is now available in all the environments of nicematrix. Before, it was available only in \texttt{\NiceArray}.
New command \texttt{\CodeAfter} (in the environments of nicematrix).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

| Symbols | \texttt{\Cell}... \texttt{\CodeAfter:n}... \texttt{\CodeAfter:_n}... \texttt{\Ddots}... \texttt{\_ddots_lines_tl}... \texttt{\Hdotsfor:...}... \texttt{\_Hdotsfor_lines_tl}... \texttt{\Hdotsfor:_n...}... \texttt{\Hdotsfor:_n...}... \texttt{\g_0_0_ddots_lines_tl}... 175, 429, 637, 638, 639, 682, 696 | \texttt{\CodeAfter:n}... \texttt{\CodeAfter:_n}... \texttt{\Ddots}... \texttt{\_ddots_lines_tl}... \texttt{\Hdotsfor:...}... \texttt{\_Hdotsfor_lines_tl}... \texttt{\Hdotsfor:_n...}... \texttt{\Hdotsfor:_n...}... \texttt{\g_0_0_ddots_lines_tl}... \texttt{\g_0_0_Hdotsfor_lines_tl}... \texttt{\g_0_0_Hdotsfor_lines_tl}... 659, 2878, 2889 | 2881, 2883 | 650, 665, 2121 | 733, 1465 | 654, 668, 2155 | 2178, 2188 | 2178, 2188 | 2178, 2188 | 2178, 2188 | \texttt{\CodeAfter:_n...}... | \texttt{\CodeAfter:_n...}... | \texttt{\CodeAfter:_n...}... | \texttt{\CodeAfter:_n...}... | \texttt{\CodeAfter:_n...}... | 648, 663, 2105 | 730, 1467 | 735, 1463, 2176, 2944 | 108 |
Contents

1  Presentation 1

2  The environments of this package 2

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

4  The PGF/Tikz nodes created by nicematrix 6

5  The code-after 7

6  The environment \NiceArray and its variants 8

7  The vertical position of the arrays 9

8  The exterior rows and columns 10

9  The dotted lines to separate rows or columns 11

10 The width of the columns 12

11 Block matrices 13