The package *nicematrix*

F. Pantigny
fpantigny@wanadoo.fr
February 7, 2020

Abstract

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

1 Presentation

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

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

- continuous dotted lines\(^1\);
- 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 `{pmatrix}` of amsmath.

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

This code composes the matrix \(A\) on the right.

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

\[
A = \begin{pmatrix} \cdots & \cdots & \cdots \\ 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots \\ 0 & \cdots & 0 \end{pmatrix}
\]

\^1This document corresponds to the version 3.11 of *nicematrix*, at the date of 2020/02/07.

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

The extension nicematrix defines the following new environments.

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

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

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

3 The continuous dotted lines

Inside the environments of the extension nicematrix, new commands are defined: \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.

\begin{bNiceMatrix}
\text{\texttt{\Ldots}} & \texttt{\Cdots} & \texttt{\Vdots} & \texttt{\Ddots} & \texttt{\Iddots}
\end{bNiceMatrix}

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

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

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

\begin{bNiceMatrix}
0 & \Cdots & 0
\end{bNiceMatrix}
In the first column of this example, there are two instructions \( \vdots \) but only one dotted line is drawn (there is no overlapping graphic objects in the resulting PDF\(^5\)).

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

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

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

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

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

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

\section{3.1 The option nullify-dots}

Consider the following matrix composed classically with the environment \{pmatrix\} of \texttt{amsmath}.

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

\[
B = \begin{pmatrix}
(a_0 & b) \\
(a_1) \\
(a_2) \\
(a_3) \\
(a_4) \\
(a_5 & b)
\end{pmatrix}
\]

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

\[
A = \begin{pmatrix}
a_0 & b \\
(a_1) \\
(a_2) \\
(a_3) \\
(a_4) \\
(a_5 & b)
\end{pmatrix}
\]

\[
B = \begin{pmatrix}
(a_0 & b) \\
(a_1) \\
(a_2) \\
(a_3) \\
(a_4) \\
(a_5 & b)
\end{pmatrix}
\]

\(^5\)And it’s not possible to draw a \( \backslash Ldots \) and a \( \backslash Cdots \) line between the same cells.

\(^6\)In \texttt{nicematrix}, one should use \( \backslash hspace* \) and not \( \backslash hspace \) 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 \{NiceArray\} (or one of its variants) with a column of type \texttt{w} or \( \texttt{W} \); see p. 10
By default, with nicematrix, if we replace \{pmatrix\} by \{pNiceMatrix\} and \vdots \text{ by } \Vdots, the geometry of the matrix is not changed.

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

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

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

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

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

### 3.2 The command \Hdotsfor

Some people commonly use the command \Hdotsfor of amsmath in order to draw horizontal dotted lines in a matrix. In the environments of nicematrix, one should use instead \Hdotsfor in order to draw dotted lines similar to the other dotted lines drawn by the package nicematrix.

As with the other commands of nicematrix (like \Cdots, \Ldots, \Vdots, etc.), the dotted line drawn with \Hdotsfor extends until the contents of the cells on both sides.

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

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

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

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

Remark: Unlike the command \Hdotsfor of amsmath, the command \Hdotsfor may be used when the extension colortbl is loaded (but you might have problem if you use \rowcolor on the same row as \Hdotsfor).
3.3 How to generate the continuous dotted lines transparently

The package \texttt{nicematrix} provides an option called \texttt{transparent} for using existing code transparently in the environments of the \texttt{amsmath} : \{\texttt{matrix}, \texttt{pmatrix}, \texttt{bmatrix}, \texttt{etc.} In fact, this option is an alias for the conjunction of two options: \texttt{renew-dots} and \texttt{renew-matrix}.\footnote{The options \texttt{renew-dots}, \texttt{renew-matrix} and \texttt{transparent} can be fixed with the command \texttt{\NiceMatrixOptions} like the other options. However, they can also be fixed as options of the command \texttt{\usepackage} (it’s an exception for these three specific options.)}

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

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

Therefore, with the option \texttt{transparent}, a classical code gives directly the output of \texttt{nicematrix}.

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

3.4 Fine tuning of the dotted lines

The distance between a node and the end of a dotted line is set by \texttt{dotted-lines-margin}. The initial value of this key is 0.3 em (it’s recommended to use a unit dependant of the current font). For an example, cf. p. 18.

4 The Tikz nodes created by \texttt{nicematrix}

The package \texttt{nicematrix} creates a Tikz node for each cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix. However, the user may wish to use directly these nodes. It’s possible. First, the user have to give a name to the array (with the key called \texttt{name}). Then, the nodes are accessible through the names “\texttt{name-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{verbatim}
\begin{pNiceMatrix}[name=mymatrix]
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pNiceMatrix}$\tikz[remember picture,overlay]
\draw (mymatrix-2-2) circle (2mm) ;
\end{verbatim}

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{verbatim}
\begin{pNiceMatrix}
\begin{pmatrix}
a & a+b & a+b+c \\
a & a & a+b \\
a & a & a
\end{pmatrix}
\end{pNiceMatrix}$
\end{verbatim}
In fact, the package \texttt{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}.\footnote{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.\footnote{There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. \pageref{foot:nosizelarge}).}

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

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}.\footnote{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.}

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

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. \pageref{nicematrix}).
5 The code-after

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

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

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

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

6 The environment \texttt{NiceArray}

The environment \texttt{NiceArray} is similar to the environment \texttt{array}. As for \texttt{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{...}{...}}, \texttt{W{...}{...}}, \texttt{|}, \texttt{>{...}}, \texttt{<{...}} and \texttt{*{n}{...}} but the letters \texttt{p}, \texttt{m} and \texttt{b} should not be used.\footnote{The column types \texttt{L}, \texttt{C} and \texttt{R} are defined locally inside \texttt{NiceArray} with \texttt{newcolumntype} of \texttt{array}. This definition overrides an eventual previous definition. In fact, the column types \texttt{w} and \texttt{v} are also redefined.}

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

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

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

In fact, there is also variants for the environment \texttt{NiceArray}: \texttt{pNiceArray}, \texttt{bNiceArray}, \texttt{BNiceArray}, \texttt{vNiceArray} and \texttt{vNiceArray}.

In the following example, we use an environment \texttt{pNiceArray} (we don’t use \texttt{pNiceMatrix} because we want to use the types \texttt{L} and \texttt{R} — in \texttt{pNiceMatrix}, all the columns are of type \texttt{C}).

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

In fact, the environment \texttt{pNiceArray} and its variants are based upon a more general environment, called \texttt{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 \texttt{NiceArrayWithDelims} if we want to use atypical delimiters.\footnote{In a command \texttt{\multicolumn}, one should also use the letters \texttt{L}, \texttt{C}, \texttt{R}.}
\begin{NiceArrayWithDelims}
\{\downarrow\downarrow\{CCC\}
1 & 2 & 3 \\downarrow
4 & 5 & 6 \\downarrow
7 & 8 & 9 \\downarrow
\end{NiceArrayWithDelims}$

7 The exterior rows and columns

The options first-row, last-row, first-col and last-col allow the composition of exterior rows and columns in the environments of nicematrix. A potential first row has the number 0 (and not 1). Idem for the potential first column. In general cases, one must specify the number of the last row and the number of the last column as values of last-row and last-col.

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

We have several remarks to do.

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

- In an environment with an explicit preamble, the option last-col must be used without value: the number of columns will be automatically computed from the preamble of the array.

- For the potential last row, the option last-row may, in fact, be used without value. In this case, nicematrix computes, during the first compilation, the number of rows of the array and writes that information in the .aux file for the second run. In the following example, the option last-row will be used without value.

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

\NiceMatrixOptions{code-for-first-row = \color{red},
code-for-first-col = \color{blue},
code-for-last-row = \color{green},
code-for-last-col = \color{magenta}}

\begin{pNiceArray}{CC|CC}
\{first-row, last-row, first-col, last-col\}
8
\end{pNiceArray}$

8
& C_1 & C_2 & C_3 & C_4 & \hline
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 \\
L_2 & a_{21} & a_{22} & a_{23} & a_{24} & L_2 \\
L_3 & a_{31} & a_{32} & a_{33} & a_{34} & L_3 \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 \\
& C_1 & C_2 & C_3 & C_4 & \end{pNiceArray}$

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

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

\begin{NiceArray}{CCCC}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{NiceArray}\right)

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 extension 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 !{\vline} 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 14.

- Logically, the potential option columns-width (described p. 10) doesn’t apply to the “first column” and “last column”.

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

8 The dotted lines to separate rows or columns

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

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
\hline
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 “|”.

\begin{NiceArray}{CCCC}
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 
\end{pNiceArray}
\[
\begin{array}{cccc}
  C_1 & C_2 & C_3 & C_4 \\
  1 & 2 & 3 & 4 & L_1 \\
  5 & 6 & 7 & 8 & L_2 \\
  9 & 10 & 11 & 12 & L_3 \\
  \hdottedline \\
  13 & 14 & 15 & 16 & L_4 \\
\end{array}
\]

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 extension \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}.

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

\section{The width of the columns}

In the environments with an explicit preamble (like \texttt{NiceArray}, \texttt{pNiceArray}, 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}.

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

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

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

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 Tikz will have written informations in the \texttt{.aux} file and a message requiring a second compilation will appear).}

\[
\begin{pNiceMatrix}[columns-width = auto]
  1 & 12 \& -123 \\
  12 \& 0 \& 0 \\
  4 \& 1 \& 2 \\
\end{pNiceMatrix}
\]
Without surprise, it’s possible to fix the width of the columns of all the matrices of a current scope with the command \NiceMatrixOptions.

\NiceMatrixOptions{columns-width=10mm}
\begin{pNiceMatrix}
  a & b \\
  c & d \\
\end{pNiceMatrix} = \begin{pNiceMatrix}
  1 & 1245 \\
  345 & 2 \\
\end{pNiceMatrix}

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

Several compilations may be necessary to achieve the job.

10 Block matrices

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

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

The command \Block must be used in the upper leftmost cell of the array with two arguments. The first argument is the size of the block with the syntax i-j where i is the number of rows of the block and j its number of columns. The second argument is the content of the block (composed in math mode). A Tikz node corresponding to the merged cells is created with the name “name-i-j” where name is the name given to the array.

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

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it’s not possible to use directly a command like \large, \Large and \LARGE. That’s why the command \Block provides an option between angle brackets to specify a TeX code which will be inserted before the beginning of the math mode.

\footnote{At this time, this is the only usage of the environment \NiceMatrixBlock but it may have other usages in the future.}
For technical reasons, you can’t write \textbackslash Block\{i-j\} {. But you can write \textbackslash Block\{i-j\} \{<\} with the expected result.

### 11 Advanced features

#### 11.1 The option small

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

```latex
\begin{bNiceArray}{CCCC|C}
\small,
last-col,
code-for-last-col = \texttt{\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 \texttt{NiceMatrix} with the option \texttt{small} is not composed exactly as the environment \texttt{smallmatrix}. Indeed, all the environments of \texttt{nicematrix} are constructed upon \texttt{array} (of the extension \texttt{array}) whereas the environment \texttt{smallmatrix} is constructed directly with an \texttt{\halign} of \TeX.

In fact, the option \texttt{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.

#### 11.2 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 col\footnote{We recall that the first row (if it exists) has the number 0 and that the first col (if it exists) has also the number 0.}. Of course, the user must not change the value of these counters which are used internally by \texttt{nicematrix}. 

\[ 
\begin{bNiceArray}{CC|C|CC}
\scriptstyle, \\
\text{L}_2 - 2 \text{L}_1 + \text{L}_2 \\
\text{L}_3 - \text{L}_1 + \text{L}_3 \\
\end{bNiceArray} \]
\begin{NiceMatrix}
[\text{first-row,} \\
\text{first-col,} \\
\text{code-for-first-row} = \mathbf{\alpha(jCol)} \\
\text{code-for-first-col} = \mathbf{\arabic(iRow)} ] \\
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{array}
\end{NiceMatrix}

\[
\begin{pmatrix}
\text{a} & \text{b} & \text{c} & \text{d} \\
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{pmatrix}
\]

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

The extension \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 \(n\) is the number of rows and \(p\) the number of columns. The second argument is the pattern (it’s a list of tokens which are inserted in each cell of the constructed matrix, excepted in the cells of the eventual exterior rows and columns).

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

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

\begin{NiceMatrix}
[light-syntax,\text{first-row,first-col}]
\begin{array}{cccc}
\alpha & \beta & \gamma & \delta \\
2\cos\alpha & \cos\alpha + \cos\beta & 2\cos\beta & \cos\alpha + \cos\beta
\end{array}
\end{NiceMatrix}

\begin{NiceMatrix}
[light-syntax,\text{first-row,first-col}]
\begin{array}{cccc}
\alpha & \beta & \gamma & \delta \\
2\cos\alpha & \cos\alpha + \cos\beta & 2\cos\beta & \cos\alpha + \cos\beta
\end{array}
\end{NiceMatrix}

11.3 The option \texttt{hlines}

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

\[
\begin{NiceMatrix}
[\text{hlines,first-row,first-col}]
\begin{array}{cccc}
\text{e} & \text{a} & \text{b} & \text{c} \\
\text{e} & \text{e} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{a} & \text{e} & \text{c} & \text{b} \\
\text{b} & \text{b} & \text{c} & \text{e} & \text{a} \\
\text{c} & \text{c} & \text{b} & \text{a} & \text{e}
\end{array}
\end{NiceMatrix}
\]

11.4 The option \texttt{light-syntax}

The option \texttt{light-syntax}\footnote{This option is inspired by the extension \texttt{spalign} of Joseph Rabinoff.} allow 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 a space 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.

\[
\begin{bNiceMatrix}
[light-syntax,\text{first-row,first-col}]
\begin{array}{cccc}
\alpha & \beta & \gamma & \delta \\
2\cos\alpha & \cos\alpha + \cos\beta & 2\cos\beta & \cos\alpha + \cos\beta
\end{array}
\end{bNiceMatrix}
\]

\[
\begin{bNiceMatrix}
[light-syntax,\text{first-row,first-col}]
\begin{array}{cccc}
\alpha & \beta & \gamma & \delta \\
2\cos\alpha & \cos\alpha + \cos\beta & 2\cos\beta & \cos\alpha + \cos\beta
\end{array}
\end{bNiceMatrix}
\]

\[
\begin{bNiceMatrix}
[light-syntax,\text{first-row,first-col}]
\begin{array}{cccc}
\alpha & \beta & \gamma & \delta \\
2\cos\alpha & \cos\alpha + \cos\beta & 2\cos\beta & \cos\alpha + \cos\beta
\end{array}
\end{bNiceMatrix}
\]
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.

11.5 Utilisation of the column type S of \texttt{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}.

\[
\begin{pNiceArray}{SC\text{Wc}{1cm}C}[\text{nullify-dots,first-row}]
\begin{array}{ccc}
C_1 & \Cdots & C_n \\
2.3 & 0 & \Cdots \\
12.4 & \Vdots & \Vdots \\
1.45 \\
7.2 & 0 & \Cdots
\end{array}
\end{pNiceArray}
\]

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

12 Technical remarks

12.1 Definition of new column types

The extension \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 \texttt{?} in order to draw a thick rule of width 1 pt. The following definition will do the job:

\[
\newcolumntype{?}{!{\OnlyMainNiceMatrix{\vrule width 1 pt}}}
\]

The thick vertical rule won’t extend in the exterior rows:

\[
\begin{pNiceArray}{CC?CC}[\text{first-row,last-row}]
C_1 & C_2 & C_3 & C_4 \\
a & b & c & d \\
e & f & g & h \\
C_1 & C_2 & C_3 & C_4
\end{pNiceArray}
\]

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

12.2 Intersections of dotted lines

Since the version 3.1 of \texttt{nicematrix}, the dotted lines created by \texttt{\Ddots}, \texttt{\Ldots}, \texttt{\Vdots}, etc. can’t intersect.\footnote{On the contrary, dotted lines created by \texttt{\hdottedline}, the letter “:” in the preamble of the array and the command \texttt{\line} in the \texttt{code-after} can have intersections with other dotted lines.}

That means that a dotted line created by one these commands automatically stops when it arrives on a dotted line already drawn. Therefore, the order in which dotted lines are drawn is important.

Here’s that order (by design) : \texttt{\Hdotsfor, \Vdots, \Ddots, \Iddots, \Cdots and \Ldots}.

With this structure, it’s possible to draw the following matrix.
The names of the Tikz nodes created by nicematrix

We have said that, when a name is given to an environment of nicematrix, it’s possible to access the Tikz nodes through this name (cf. p. 5). That’s the recommended way to access these nodes. However, we describe now the internal names of these nodes.

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

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

Diagonal lines

By default, all the diagonal lines\(^{19} \) of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That’s why the position of the instructions \( \Ddots \) in the array can have a marked effect on the final result.

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

Example with parallelization (default):

\[
\begin{pNiceMatrix}
1 & \Ddots & & 1 \\
\vdots & \Ddots & \vdots & \vdots \\
\end{pNiceMatrix}
\]

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

The same example without parallelization:

\[
\begin{pNiceMatrix}
1 & \Ddots & & 1 \\
\vdots & \Ddots & \vdots & \vdots \\
\end{pNiceMatrix}
\]

\(^{19}\)We speak of the lines created by \( \Ddots \) and not the lines created by a command \line in code-after.
12.5 The “empty” cells

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

For \nicematrix, the precise rules are as follow.

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

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

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

- Each cell whose TeX output has a width less than 0.5 \text{pt} is empty.

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

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

12.6 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 extension \nicematrix does the same in all its environments, \{NiceArray\} included. However, if the user wants the environment \{NiceArray\} behaving by default like the environment \{array\} of \array (for example, when adapting an existing document) it’s possible to control this behaviour with the option exterior-arraycolsep, set by the command \NiceMatrixOptions. With this option, exterior spaces of length \arraycolsep will be inserted in the environments \{NiceArray\} (the other environments of \nicematrix are not affected).

12.7 The class option draft

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

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

\footnote{The main reason is that we want dotted lines with round dots (and not square dots) with the same space on both extremities of the lines. To achieve this goal, we have to construct our own system of dotted lines.}
12.8 A technical problem with the argument of $\\$

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

\begin{pNiceMatrix}
  a & \frac{AB}{} \\
  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.

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

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

\begin{pNiceMatrix}
  a & \multicolumn1C{\frac{AB}{}} \\
  b & c
\end{pNiceMatrix}

12.9 Obsolete environments

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

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

Since the version 3.8, an error is raised when one of these environments is used. It’s still possible to use these environments by loading nicematrix with the option obsolete-environments. However, these environments will probably be completely deleted in a future version of nicematrix.

13 Examples

13.1 Dotted lines

A tridiagonal matrix:

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

A permutation matrix (as an example, we have raised the value of \textit{dotted-lines-margin}).

\begin{pNiceMatrix}[dotted-lines-margin=0.6em]
\begin{array}{cccccccccc}
0 & 1 & 0 & & \Cdots & 0 \\
\Vdots & & & \Ddots & & \Vdots \\
& & & \Ddots & & \\
& & & \Ddots & & 0 \\
0 & 0 & & & & 1 \\
1 & 0 & & \Cdots & & 0 \\
\end{array}
\end{pNiceMatrix}

An example with \texttt{\textbackslash idots}:

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

An example with \texttt{\textbackslash 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 \texttt{\textbackslash 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 \\
0 & 1 & 1 & 1 & 1 & 0 \\
\end{pNiceMatrix}
An example for the resultant of two polynomials:

\[
\begin{vNiceArray}{CCCC:CCC}[columns-width=6mm]
\begin{array}{c}
\begin{array}{cccc}
a_0 & \ & \ & b_0 \\
a_1 & \ & \ & b_1 \\
\vdots & \ & \ & \vdots \\
a_p & \ & a_0 & \ \\
& & & & b_1 \\
& & & & \vdots \\
& & & & a_p \\
& & b_0 & b_1 \end{array}
\end{array}
\end{vNiceArray}
\]

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

$$
\begin{pNiceArray}{*6C|C}[nullify-dots,last-col,code-for-last-col={\scriptstyle}]
\begin{array}{cccccc}
1 & 1 & 1 & \cdots & 1 & 0 \\
0 & 1 & 0 & \cdots & 0 & \ \text{L}_2 \gets \text{L}_2 - \text{L}_1 \\
0 & 0 & 1 & \cdots & \vdots & \ \text{L}_3 \gets \text{L}_3 - \text{L}_1 \\
& & & \cdots & \vdots & \vdots \\
\vdots & & & & 0 & \ \vdots \\
0 & & & & 1 & \ \vdots \\
& & & & b_0 & b_1 \\
& & & & b_q & b_1 \\
& & & & \vdots & \vdots \\
& & & & a_p & b_q
\end{array}
\end{pNiceArray}
$$

Matrix width in the following example:

\[
\begin{array}{ccccccc}
1 & 1 & 1 & \cdots & 1 & \cdots & 1 \\
0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & \vdots & \cdots & \vdots \\
\vdots & \vdots & \vdots & \cdots & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
& & & & & \text{L}_{n-1} - \text{L}_n - \text{L}_1
\end{array}
\]

13.2 Width of the columns

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

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

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

```latex
\begin{pNiceArray}{>{\strut}CCCC}
\begin{tikzpicture}
\[name suffix = -large,
\end{tikzpicture}
\end{pNiceArray}
```

Now, let’s see how to color the cells.
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`.

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

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.
\begin{Verbatim}
\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
\end{Verbatim}

We recall that, for a rectangle of merged cells (with the command \textbackslash Block), a Tikz node is created for the set of merged cells with the name $i$-$j$ where $i$ and $j$ are the number of the row and the number of the column of the upper left cell (where the command \textbackslash Block has been issued).

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

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

\begin{Verbatim}
\begin{pNiceArray}{CCC}\
\[\text{name=example, last-col, create-medium-nodes}\
\begin{block}{3-3}<\Large>{A} & & & 0 \\
0 & \hspace*{1cm} & & \Vdots \\
0 & & & 0 \\
\end{block}
\end{pNiceArray}
\end{Verbatim}
If we want to highlight each row of this matrix, we can use the previous technique three times.

\begin{tikzpicture}[mes-options]
\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.

\[
\begin{pmatrix}
(a & a+b & a+b+c)_{L_1} \\
(a & a & a+b)_{L_2} \\
(a & a & a)_{L_3}
\end{pmatrix}
\]

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.

\[
\begin{pmatrix}
(a & a+b & a+b+c)_{L_1} \\
(a & a & a+b)_{L_2} \\
(a & a & a)_{L_3}
\end{pmatrix}
\]

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

\begin{pNiceArray}{>{\strut}CCCC}
\[create-large-nodes,margin,extra-margin=2pt,
\text{code-after} = \{\tikz \path [name suffix = -large,
fill = red!15,
blend mode = multiply]\]
\hline
A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & A_{22} & A_{23} & A_{24} \\
\end{pNiceArray}
\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions{nullify-dots}
\begin{array}{cc}
\text{The matrix } B \text{ has a “first row” (for } C_j \text{) and that’s why we use the key } \text{first-row}. \\
\begin{bNiceArray}{C>{\strut}CCCC}[name=B,first-row]
& & C_j \\
\b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
\Vdots & & \Vdots & & \Vdots \\
& & b_{kj} \\
& & \Vdots \\
\b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn}
\end{bNiceArray}
\begin{array}{cc}
\text{The matrix } A \text{ has a “first column” (for } L_i \text{) and that’s why we use the key } \text{first-col}. \\
\begin{bNiceArray}{CC>{\strut}CCC}[name=A,first-col]
& a_{11} & \Cdots & & & a_{1n} \\
& \Vdots & & & & \Vdots \\
L_i & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} \\
& \Vdots & & & & \Vdots \\
& a_{n1} & \Cdots & & & a_{nn}
\end{bNiceArray}
\end{array}
\end{array}
\end{NiceMatrixBlock}

In the matrix product, the two dotted lines have an open extremity.
\begin{bNiceArray}{CC>{\strut}CCC}[name=C]
& & & & \\
& & \Vdots \\
\Cdots & & c_{ij} \\
\end{bNiceArray}
\end{NiceMatrixBlock}

### 13.4 Direct utilisation of the Tikz nodes

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

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

\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions{nullify-dots}
\begin{array}{cc}
\text{The three matrices will be displayed using an environment } \{array\} \text{ (an environment } \{tabular\} \text{ may also be possible).}
\end{array}
\end{NiceMatrixBlock}
14 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`.

14.1 Declaration of the package and extensions loaded

```latex
\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}
```

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

```latex
\begin{verbatim}
\RequirePackage{tikz}
\usetikzlibrary{fit}
\RequirePackage{expl3}[2019/07/01]
\end{verbatim}
```

We test if the class option `draft` has been used. In this case, we raise the flag `\_@@_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
```

22cf. tex.stackexchange.com/questions/57424/using-of-usetikzlibrary-in-an-expl3-package-fails
The command for the treatment of the options of `\usepackage` is at the end of this package for technical reasons.

We load `array` and `amsmath`.

\begin{verbatim}
\RequirePackage { array }
\RequirePackage { amsmath }
\RequirePackage { xparse } [ 2018-07-01 ]
\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:nn { 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 } }
\end{verbatim}

### 14.2 Technical definitions

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

\begin{verbatim}
\bool_new:N \c_@@_revtex_bool
\@ifclassloaded { revtex4-1 } { \bool_set_true:N \c_@@_revtex_bool }
\@ifclassloaded { revtex4-2 } { \bool_set_true:N \c_@@_revtex_bool }
\end{verbatim}

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

\begin{verbatim}
\@@_msg_new:nn { Draft~mode }
{ The~compilation~is~in~draft~mode:~the~dotted~lines~won't~be~drawn. }
\bool_if:NT \c_@@_draft_bool
{ \msg_warning:nn { nicematrix } { Draft~mode } }
\end{verbatim}

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.

\begin{verbatim}
\ProvideDocumentCommand \iddots { }{
  \mathinner
  { \mkern 1 mu \raise \p@ 
    \hbox:n { . } \mkern 2 mu
    \raise 4 \p@ \hbox:n { . } \mkern 2 mu
    \raise 7 \p@ \vbox { \kern 7 pt \hbox:n { . } } \mkern 1 mu}
}\end{verbatim}

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.

\begin{verbatim}
\int_new:N \g_@@_env_int
\end{verbatim}
We also define a counter to count the environments \{NiceMatrixBlock\}.
\begin{verbatim}
\int_new:N \g_@@_NiceMatrixBlock_int
\end{verbatim}

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 \texttt{columns-width} is used with the special value \texttt{auto}, the boolean \l_@@_auto_columns_width_bool also will be raised).
\begin{verbatim}
\dim_new:N \l_@@_columns_width_dim
\end{verbatim}

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

We want to know if we are in an environment of \texttt{nicematrix} because we will raise an error if the user tries to use nested environments.
\begin{verbatim}
\bool_new:N \l_@@_in_env_bool
\end{verbatim}

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

Consider the following code:
\begin{verbatim}
$\begin{pNiceMatrix}
a & b & c \\
d & e & \Vdots \\
f & \Cdots \\
g & h & i \\
\end{pNiceMatrix}$
\end{verbatim}

First, the dotted line created by the \texttt{\Vdots} will be drawn. The implicit cell in position 2-3 will be considered as “dotted”. Then, we will have to draw the dotted line specified by the \texttt{\Cdots}; the final extremity of that line will be exactly in position 2-3 and, for that new second line, it should be considered as a \textit{closed} extremity (since it is dotted). However, we don’t have the (normal) Tikz node of that node (since it’s an implicit cell): we can’t draw such a line. That’s why that dotted line will be said \textit{impossible} and an error will be raised.\footnote{Of course, the user should solve the problem by adding the lacking ampersands.}
\begin{verbatim}
\bool_new:N \l_@@_impossible_line_bool
\end{verbatim}

We have to know whether \texttt{colortbl} is loaded for the redefinition of \texttt{\everycr} and for \texttt{\vline}.
\begin{verbatim}
\bool_new:N \c_@@_colortbl_loaded_bool
\AtBeginDocument
{ \ifpackageloaded { colortbl } \{ \bool_set_true:N \c_@@_colortbl_loaded_bool \cs_set_protected:Npn \@@_vline_i: { \CT@arc@ \vline } \} \} \end{verbatim}
The length \texttt{\l@@inter_dots_dim} is the distance between two dots for the dotted lines. The default value is 0.45 em but it will be changed if the option \texttt{small} is used.
\begin{verbatim}
\dim_new:N \l@@inter_dots_dim
\dim_set:Nn \l@@inter_dots_dim { 0.45 \text{ em} }
\end{verbatim}

The length \texttt{\l@@dotted_lines_margin_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).
\begin{verbatim}
\dim_new:N \l@@dotted_lines_margin_dim
\dim_set:Nn \l@@dotted_lines_margin_dim { 0.3 \text{ em} }
\end{verbatim}

The length \texttt{\l@@radius_dim} is the radius of the dots for the dotted lines. The default value is 0.34 pt but it will be changed if the option \texttt{small} is used.
\begin{verbatim}
\dim_new:N \l@@radius_dim
\dim_set:Nn \l@@radius_dim { 0.53 \text{ pt} }
\end{verbatim}

The name of the current environment or the current command (will be used only in the error messages).
\begin{verbatim}
\str_new:N \g@@type_env_str
\tl_new:N \g@@code_after_tl
\end{verbatim}

The counters \texttt{\l@@save_iRow_int} and \texttt{\l@@save_jCol_int} will be used to save the values of the eventual LaTeX counters \texttt{iRow} and \texttt{jCol}. These LaTeX counters will be restored at the end of the environment.
\begin{verbatim}
\int_new:N \l@@save_iRow_int
\int_new:N \l@@save_jCol_int
\end{verbatim}

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

14.2.1 Variables for the exterior rows and columns

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

- **First row**
  The integer \texttt{\l@@first_row_int} is the number of the first row of the array. The default value is 1, but, if the option \texttt{first-row} is used, the value will be 0. As usual, the global version is for the passage in the \texttt{\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 \texttt{\l@@first_col_int} is the number of the first column of the array. The default value is 1, but, if the option \texttt{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}

27
• **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}

• **Last column**
  For the eventual “last column”, we use an integer. A value of −1 means that there is no last column.

  \begin{verbatim}
  \int_new:N \l_@@_last_col_int
  \int_set:Nn \l_@@_last_col_int { -1 }
  \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}

  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.

  \begin{verbatim}
  \bool_new:N \g_@@_last_col_found_bool
  \end{verbatim}

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

14.2.2 The column S of siunitx

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

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

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

\footnote{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.}
\renewcommand*{\NC@rewrite@S}{\[1\][1]}
{
\@temptokena \exp_after:wN
\{ \tex_the:D \@temptokena
> \{ \_\_\_\_\_\_\_\_\siunitx_table_collect_begin: S {#1} \}
\}
\NC@find
}\}

We want to patch this command (in the environments of nicematrix) in order to have:
\renewcommand*{\NC@rewrite@S}{\[1\][1]}
{
\@temptokena \exp_after:wN
\{ \tex_the:D \@temptokena
> \{ \_\_\_\_\_\_\_\_\_\_\siunitx_table_collect_begin: S {#1} \}
\}
\NC@find
}\}

However, we don’t want do use explicitly any private command of siunitx. That’s why we will extract the name of the two \_\_\_\_\_\_\_\_\_\_\siunitx... commands by their position in the code of \NC@rewrite@S. Since the command \NC@rewrite@S appends some tokens to the \@temptokena list, we use the LaT\eX command \NC@rewrite@S in a group (\group_begin: – \group_end:) and we extract the two commands which are in the \@temptokena tokens. However, this extraction can be done only when siunitx is loaded (and it may be loaded after nicematrix) and, in fact, after the beginning of the document — because some instructions of siunitx are executed in a \AtBeginDocument. That’s why this extraction will be done only at the first utilisation of an environment of nicematrix with the command \@@_adapt_S_column:
\cs_set_protected:Npn \@@_adapt_S_column:
101 \bool_if:NT \c_@@_siunitx_loaded_bool
102 { \group_begin:
103 \@temptokena = { }
104 \cs_set_eq:NN \NC@find \prg_do_nothing:
105 \NC@rewrite@S { }
\group_end:
106 \tl_new:N \c_@@_table_collect_begin_tl
107 \tl_gset:Nx \c_@@_table_collect_begin_tl { \tl_item:Nn \g_tmpa_tl 2 }
108 \tl_new:N \c_@@_table_print_tl
109 \tl_gset:Nx \c_@@_table_print_tl { \tl_item:Nn \g_tmpa_tl -1 }
110 \cs_set_protected:Npn \@@_adapt_S_column:

In the preamble of the LaT\eX document, the boolean \c_@@_siunitx_loaded_bool won’t be known. That’s why we test the existence of \c_@@_siunitx_loaded_bool and not its value.\footnote{Indeed, nicematrix may be used in the preamble of the LaT\eX document. For example, in this document, we compose a matrix in the box \ExampleOne before loading arydshln (because arydshln is not totally compatible with nicematrix).}
101 \bool_if:NT \c_@@_siunitx_loaded_bool
102 { \group_begin:
103 \@temptokena = { }
104 \cs_set_eq:NN \NC@find \prg_do_nothing:
105 \NC@rewrite@S { }
\group_end:
Conversion of the \@temptokena tokens in a token list of expl3 (the tokens 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 \c_@@_adapt_S_column: becomes no-op (globally).
\begin{verbatim}
\cs_gset_eq:NN \@@_adapt_S_column: \prg_do_nothing:
\end{verbatim}

The command \c_@@_renew_NC@rewrite@S: will be used in each environment of \nicematrix in order to “rewrite” the S column in each environment (only if the boolean \c_@@_siunitx_loaded_bool is raised, of course).
\begin{verbatim}
\cs_new_protected:Npn \@@_renew_NC@rewrite@S:
\begin{verbatim}
\renewcommand*{\NC@rewrite@S}{\exp_after:wN \@temptokena \exp_after:wN \ter_the:D \@temptokena > { \@@_Cell: \c_@@_table_collect_begin_tl S {##1} } c < { \c_@@_table_print_tl \@@_end_Cell: } }
\NC@find
\end{verbatim}
\end{verbatim}

14.3 The options

The boolean \l_@@_light_syntax_bool corresponds to the option light-syntax.
\begin{verbatim}
\bool_new:N \l_@@_light_syntax_bool
\end{verbatim}

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

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

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

\bool_new:N \l_@@_auto_columns_width_bool

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

\str_new:N \l_@@_name_str

The boolean \l_@@_extra_medium_bool will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

\bool_new:N \l_@@_medium_nodes_bool
\bool_new:N \g_@@_medium_nodes_bool
\bool_new:N \l_@@_large_nodes_bool
\bool_new:N \g_@@_large_nodes_bool

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

\dim_new:N \l_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim
\dim_new:N \g_@@_width_last_col_dim
\dim_new:N \g_@@_width_first_col_dim

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

\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim
\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim

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.

\tl_new:N \l_@@_end_of_row_tl
\tl_set:Nn \l_@@_end_of_row_tl { ; }

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

\keys_define:nn { NiceMatrix / Global }

{  
 damped-lines-margin .dim_set:N = \l_@@_dotted_lines_margin_dim ,
 damped-lines-margin .value_required:n = true ,
 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-first-row .tl_set:N = \l_@@_code_for_last_row_tl ,
 code-for-first-row .value_required:n = true ,
 code-for-first-row .tl_set:N = \l_@@_code_for_last_row_tl ,
 code-for-first-row .value_required:n = true ,
 code-for-first-row .tl_set:N = \l_@@_code_for_last_row_tl ,
 code-for-first-row .value_required:n = true ,
 small .bool_set:N = \l_@@_small_bool ,
 hlines .bool_set:N = \l_@@_hlines_bool ,
 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
}

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

renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
renew-dots .value_forbidden:n = true ,
nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
In some circumstances, the “medium nodes” are created automatically, for example when a dotted line has an “open” extremity (idem for the “large nodes”).

```latex
create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
create-extra-nodes .meta:n =
  { create-medium-nodes , create-large-nodes } ,
left-margin .dim_set:N = \l_@@_left_margin_dim ,
left-margin .default:n = \arraycolsep ,
right-margin .dim_set:N = \l_@@_right_margin_dim ,
right-margin .default:n = \arraycolsep ,
margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
margin .default:n = \arraycolsep ,
extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
extra-margin .meta:n =
  { extra-left-margin = #1 , extra-right-margin = #1 }
```

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

```latex
\keys_define:nn { NiceMatrix / Env }
{
  columns-width .code:n =
  \str_if_eq:nnTF { #1 } { auto }
  { \bool_set_true:N \l_@@_auto_columns_width_bool } ,
  columns-width .value_required:n = true ,
  name .code:n = \unless \ifmeasuring@
  \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
  { \@@_error:nn { Duplicate~name } { #1 } } ,
  name .value_required:n = true ,
  code-after .tl_gset:N = \g_@@_code_after_tl ,
  first-col .code:n = \int_zero:N \l_@@_first_col_int ,
  first-row .code:n = \int_zero:N \l_@@_first_row_int ,
  last-row .int_set:N = \l_@@_last_row_int ,
  last-row .default:n = -1 ,
}
```

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

```latex
\keys_define:nn { NiceMatrix }
{
  NiceMatrixOptions .inherit:n =
  { NiceMatrix / Global ,
  } ,
  NiceMatrix .inherit:n =
  { NiceMatrix / Global ,
    NiceMatrix / Env
  } ,
  NiceArray .inherit:n =
  { NiceMatrix / Global ,
    NiceMatrix / Env
  } ,
```
We finalise the definition of the set of keys “NiceMatrix / NiceMatrixOptions” with the options specific to \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.

\keys_define:nn { NiceMatrix / NiceMatrixOptions }
{ }

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

If the option columns-width is used, all the columns will have the same width.

If \texttt{\NiceMatrixOptions}, the special value auto is not available.

By default, the specifier used in the preamble of the array (for example in \texttt{\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).

```
\def\str_new:N \l_@@_letter_for_dotted_lines_str
\def\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

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

```
\@@_msg_redirect_name:nn { Duplicate-name } { none } ,
```

```
\_msg_redirect_name:nn { Duplicate-name } { none } ,
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

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

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\def\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\str_new:N \l_@@_letter_for_dotted_lines_str
```

```
\@@_error:n { Bad-value-for-letter-for-dotted-lines } ,
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\def\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```

```
\int_compare:nTF { \tl_count:n { #1 } = \c_one_int } { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } } { \@@_error:n { Bad-value-for-letter-for-dotted-lines } } ,
```

```
\str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str
```
\NiceMatrixOptions is the command of the nicematrix package to fix options at the document level. The scope of these specifications is the current TeX group.

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

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

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

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

14.4 Important code used by \{NiceArrayWithDelims\}

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

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

\int_gincr:N \c@jCol
Now, we increment the counter of the rows. We don’t do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don’t want to take into account.

\begin{verbatim}
\int_compare:nNnT \c@c@jCol = \c@c_one_int
{ \int_compare:nNnT \l@_@@_first_col_int = \c_one_int
  \@@_begin_of_row:
  \int_gset:Nn \g@@_col_total_int
  { \int_max:nn \g@@_col_total_int \c@c@jCol }
}
\end{verbatim}

The content of the cell is composed in the box `\l@tmpa_box` because we want to compute some dimensions of the box. The `\box_set_end:` corresponding to this `\hbox_set:Nw` will be in the `\@@_end_Cell:` (and the `\c_math_toggle_token` also).

\begin{verbatim}
\box_set:Nw \l@tmpa_box
\c_math_toggle_token
\bool_if:NT \l@@_small_bool \scriptstyle
\end{verbatim}

We will call `corners` of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn’t always exist simultaneously). The codes `\l@_@@_code_for_first_row_tl` and `al` don’t apply in the corners of the matrix.

\begin{verbatim}
\int_compare:nNnTF \c@iRow = 0
{ \int_compare:nNnT \c@jCol > 0 \l@_@@_code_for_first_row_tl }
{ \int_compare:nNnT \c@iRow = \l@@_last_row_int
  \l@_@@_code_for_last_row_tl }
}
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Nn \@@_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 }
}
\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.

\begin{verbatim}
\cs_new_protected:Np \@@_actualization_for_first_and_last_row:
{ \int_compare:nNnT \c@iRow = 0
  \dim_gset:Nn \g@@_dp_row_zero_dim
  { \dim_max:nn \g@@_dp_row_zero_dim \l@tmpb_box } }
\end{verbatim}
We want to compute in \( g_{\text{max cell width dim}} \) the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

\[
\dim_gset:Nn \ g_{\text{max cell width dim}}
\{ \dim_max:nn \ g_{\text{max cell width dim}} { \box_wd:N \ l_{\text{tmpa box}} } \}
\]

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

\( \mathcal{O} \text{actualization for first and last row:} \)

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

\[
\text{tikz}
\[
\text{remember-picture ,}
\text{inner-sep = c.zero_dim ,}
\text{minimum-width = c.zero_dim ,}
\text{baseline}
\]
\[
\text{node }
\[
\text{anchor = base ,}
\text{name = nm - \int_use:N \ g_{\text{env int}} -}
\text{\int_use:N \ c@iRow -}
\text{\int_use:N \ c@jCol ,}
\text{alias =}
\text{\str_if_empty:NF \ l_{\text{name str}}}
\}
\[
\text{\l_{\text{name str}} -}
\text{\int_use:N \ c@iRow -}
\text{\int_use:N \ c@jCol}
\]
\[
\text{\bgroup}
\text{\box_use:N \ l_{\text{tmpa box}}}
\text{\egroup ;}
\]
\[
\text{\cs_generate_variant:Nn \ dim_set:Nn \ { N x } }
\]

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

\[
\text{cs_new_protected:Nn \ \@\_renewcolumntype:nn}
\[
\{ \newcolumntype #1 [ 2 ] 
\{ 
\text{> {}
\text{\hspace*{\c_zero_dim} \ l_{\text{tmpa box}}}
\text{\@@_Cell:}
\}
\text{c}
\text{< {}
\text{\\@\_end_Cell:}
\text{\\box_set_end:}
\text{#2}
\text{\tikz [ remember-picture ]}
\text{\node}
\]
\]
The argument of the following command \@@_instruction_of_type:n defined below is the type of the instruction (Cdots, Vdots, Ddots, etc.). This command writes in the corresponding \g_@@_type_lines_tl the instruction which will really draw the line after the construction of the matrix.

For example, for the following matrix,
\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots \\
\end{pNiceMatrix}
the content of \g_@@_Cdots_lines_tl will be:
\@@_draw_Cdots:nn {2}{2}
\@@_draw_Cdots:nn {3}{2}

We begin with a test of the flag \c_@@_draft_bool because, if the key draft is used, the dotted lines are not drawn.
\bool_if:NTF \c_@@_draft_bool
\cs_set_protected:Npn \@@_instruction_of_type:n #1 { }
\cs_new_protected:Npn \@@_instruction_of_type:n #1
\{ 
It’s important to use a \tl_gput_right:cx and not a \tl_gput_left:cx because we want the \Ddots lines to be drawn in the order of appearance in the array (for parallelisation).
\tl_gput_right:cx
\{ \@@_ #1 _ lines _ tl \}
\use:c { \@@_draw _ #1 : nn }
\{ \int_use:N \c@iRow \}
\{ \int_use:N \c@jCol \}
\}
\}

We want to use \array of array. However, if the class used is revtex4-1 or revtex4-2, we have to do some tuning and use the command \@array@array instead of \array because these classes do a redefinition of \array incompatible with our use of \array.
\cs_new_protected:Npn \@@_array:
\{ 
\bool_if:NTF \c_@@_revtex_bool
\cs_set_eq:NN \acoll \@arrayacoll
\}
\}

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots \\
\end{pNiceMatrix}
\cs_set_eq:NN \acolr \arrayacol
\cs_set_eq:NN \acol \arrayacol
\cs_set:Npn \halignto { }
\@@everycr:
{ \noalign { \@@everycr_i: } }
\cs_new_protected:Npn \@@_pre_array:
{\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

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

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

We switch to a global version of the \l_@@_medium_nodes_bool and \l_@@_large_nodes_bool because these booleans may be raised in cells of the array (for example in commands \Block).
\bool_gset_eq:NN \g_@@_medium_nodes_bool \l_@@_medium_nodes_bool
\bool_gset_eq:NN \g_@@_large_nodes_bool \l_@@_large_nodes_bool
The environment \{array\} uses internally the command \ialign. We change the definition of \ialign for several reasons. In particular, \ialign sets \everycr to \{ \} and we need to have to change the value of \everycr.

\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 { \@@_everycr: }
\tabskip = \c_zero_skip}
\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 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.

\begin{verbatim}
\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_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 }
\end{verbatim}

After its first utilisation, the definition of \ialign will revert automatically to its default definition. With this programmation, we will have, in the cells of the array, a clean version of \ialign.27

\begin{verbatim}
\cs_set:Npn \ialign
{\everycr { \} \tabskip = \c_zero_skip \halign}
\end{verbatim}

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\}.

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

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

27 The user will probably not employ directly \ialign in the array... but more likely environments that utilize \ialign internally (e.g.: \{substack\}).
\cs_set_eq:NN \Block \@@_Block:
\cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
\bool_if:NT \l_@@_renew_dots_bool
  \begin{itemize}
    \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 \ddots \@@_Ddots
    \item \cs_set_eq:NN \iddots \@@_Iddots
    \item \cs_set_eq:NN \ldots \@@_Ldots
    \item \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
  \end{itemize}

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

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

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

\int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

At the end of the environment \texttt{\{array\}}, \texttt{\c@iRow} will be the total number of rows. \texttt{\g_@@_row_total_int} will be the number of rows excepted the last row (if \texttt{\l_@@_last_row_bool} has been raised with the option \texttt{last-row}).

\int_gzero_new:N \g_@@_row_total_int

The counter \texttt{\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 \texttt{\g_@@_col_total_int}. These counters are updated in the command \texttt{\@@_Cell:} executed at the beginning of each cell.

\int_gzero_new:N \g_@@_col_total_int
\cs_set_eq:NN \@ifnextchar \new@ifnextchar
\cs_set_eq:NN \NC@find@w \relax
\cs_set_eq:NN \NC@find@W \relax
\@@_renewcolumntype:nn w { }
\@@_renewcolumntype:nn W { \cs_set_eq:NN \hss \hfil }

By default, the letter used to specify a dotted line in the preamble of an environment of \texttt{nicematrix} (for example in \texttt{\{pNiceArray\}}) is the letter \texttt{\_}. However, this letter is used by some extensions, for example \texttt{arydshln}. That’s why it’s possible to change the letter used by \texttt{nicematrix} with the option \texttt{letter-for-dotted-lines} which changes the value of \texttt{\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).

\tl_set_rescan:Nno
\l_@@_letter_for_dotted_lines_str \l_@@_letter_for_dotted_lines_str
\exp_args:NV \newcolumntype \l_@@_letter_for_dotted_lines_str

We nullify the definitions of the column types \texttt{w} and \texttt{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 \texttt{\relax} and not \texttt{\prg_do_nothing:}.

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

If the array is an array with all the columns of the same width, we don’t ask for the creation of the extra nodes because we will use the “\texttt{col}” nodes for the vertical dotted line.

\bool_if:nF
\begin{itemize}
  \item \l_@@_auto_columns_width_bool
    \| \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim
\end{itemize}

40
Consider the following code:
\begin{NiceArray}{C:CC:C}
a & b \\
c & d \\
e & f & g & h \\
i & j & k & l
\end{NiceArray}

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

\int_compare:nNnT \c@jCol > \g_@@_last_vdotted_col_int
{ \int_gset_eq:NN \g_@@_last_vdotted_col_int \c@jCol
\tl_gput_right:Nx \g_@@_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:
\int_gset:Nn \g_@@_last_vdotted_col_int { -1 }
\bool_gset_false:N \g_@@_last_col_found_bool

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

\tl_gclear_new:N \g_@@_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
}

14.5 The environment \{NiceArrayWithDelims\}

\NewDocumentEnvironment { NiceArrayWithDelims } { m m O { } m ! O { } }
{ \str_if_empty:NT \g_@@_type_env_str
{ \str_gset:Nn \g_@@_type_env_str { \{ \}
{ environment ~ { NiceArrayWithDelims } }
} }
\@@_adapt_S_column:
\@@_test_if_math_mode:
\bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
\bool_set_true:N \l_@@_in_env_bool
\cs_if_exist:NT \tikz@library@external@loaded
{ \tikzset { external / export = false }
}

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

\cs_if_exist:NT \tikz@library@external@loaded
{ \tikzset { external / export = false } }
\cs_if_exist:NT \ifstandalone
  \{ \tikzset { external / optimize = false } \}
\}

We increment the counter \g_@@_env_int which counts the environments of the extension.
\int_gincr:N \g_@@_env_int
\bool_if:NF \l_@@_block_auto_columns_width_bool
  \{ \dim_gzero_new:N \g_@@_max_cell_width_dim \}

We do a redefinition of \arrayrule because we want that the vertical rules drawn by | in the preamble of the array don’t extend in the potential exterior rows.
\cs_set_protected:Npn \arrayrule
  \{ \addtopreamble \@@_vline: \}

The set of keys is not exactly the same for \NiceArray and for the variants of \NiceArray (\pNiceArray, \bNiceArray, etc.) because, for \NiceArray, we have the options t, c and b.
\bool_if:NTF \l_@@_NiceArray_bool
  \{ \keys_set:nn { NiceMatrix / \NiceArray } \}
  \{ \keys_set:nn { NiceMatrix / \pNiceArray } \}
  \{ \keys_set:nn { NiceMatrix / \bNiceArray } \}
  \{ #3 \, #5 \}

A value of \(-1\) for the counter \l_@@_last_row_int means that the user has used the option last-row without value, that is to say without specifying the number of that last row. In this case, we try to read that value from the aux file (if it has been written on a previous run).
\int_compare:nNnT \l_@@_last_row_int = { -1 }
  \{ \bool_set_true:N \l_@@_last_row_without_value_bool \}

A value based on the name is more reliable than a value based on the number of the environment.
\str_if_empty:NT \l_@@_name_str
  \{ \cs_if_exist:cT { \@@_last_row_ \int_use:N \g_@@_env_int } \}
  \{ \int_set:Nn \l_@@_last_row_int \{ \use:c { \@@_last_row_ \int_use:N \g_@@_env_int } \} \}
\}
\}
\cs_if_exist:cT { \@@_last_row_ \l_@@_name_str }
  \{ \int_set:Nn \l_@@_last_row_int \{ \use:c { \@@_last_row_ \l_@@_name_str } \} \}
\}

The code in \@@_pre_array: is used only by \NiceArrayWithDelims. It exists only for historical reasons. We should change that.
\@@_pre_array:

We compute the width of the two delimiters.
\dim_gzero_new:N \g_@@_left_delim_dim
\dim_gzero_new:N \g_@@_right_delim_dim
\bool_if:NTF \l_@@_NiceArray_bool
  \{ \dim_gset:Nn \g_@@_left_delim_dim \{ 2 \arraycolsep \} \}
  \{ \dim_gset:Nn \g_@@_right_delim_dim \{ 2 \arraycolsep \} \}
\group_begin:
\dim_set_eq:NN \nulldelimiterspace \c_zero_dim
\hbox_set:Nn \l_tmpa_box \{ \c_math_toggle_token \left #1 \vcenter to 3 cm { } \right. \}
\c_math_toggle_token \}
\dim_gset:Nn \g_@@_left_delim_dim \{ \box_wd:N \l_tmpa_box \}
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.

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).

Now, the box containing the array is constructed (in \l_@@_the_array_box). However, we have some computations to do before inserting that box in the TeX flow (mainly because the exterior columns are in overlapping positions).
\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, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in \g_@@_width_first_col_dim: see p. 48).

\int_compare:nNnT \l_@@_first_col_int = 0
\{ \\
\skip_horizontal:n \arraycolsep \\
\skip_horizontal:n \g_@@_width_first_col_dim
\}

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

\bool_if:NTF \l_@@_NiceArray_bool
\{ \\
\int_compare:nNnTF \l_@@_first_row_int = 0
\{ \\
\str_if_eq:VnT \l_@@_pos_env_str { t }
\{ \\
\box_move_up:nn { \g_@@_dp_row_zero_dim + \g_@@_ht_row_one_dim }
\}
\}
\}
\int_compare:nNnT \l_@@_last_row_int > 0
\{ \\
\str_if_eq:VnT \l_@@_pos_env_str { b }
\{ \\
\box_move_down:nn { \g_@@_ht_last_row_dim + \g_@@_dp_ante_last_row_dim }
\}
\}
\}
\box_use_drop:N \l_@@_the_array_box 
\}

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:nNnT \l_@@_first_row_int = 0
\{ \\
\dim_set:Nn \l_tmpa_dim \{ \g_@@_dp_row_zero_dim + \g_@@_ht_row_zero_dim \}
\}
\dim_zero:N \l_tmpa_dim 

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:nNnT \l_@@_last_row_int > \{-2 \}
\{ \\
\dim_set:Nn \l_tmpb_dim \{ \g_@@_ht_last_row_dim + \g_@@_dp_last_row_dim \}
\}
\dim_zero:N \l_tmpb_dim 
\hbox_set:Nn \l_tmpa_box 
\}

A value of −1 for \l_@@_last_row_int means that there is a “last row” but the number of that row is unknown (the user have not set the value with the option last row).
We take into account the “first row” (we have previously computed its total height in \l_tmpa_dim).

\skip_vertical:n { - \l_tmpa_dim }
\hbox:n
{ \skip_horizontal:n { - \arraycolsep }
\box_use_drop:N \l_@@_the_array_box
\skip_horizontal:n { - \arraycolsep }
}

We take into account the “last row” (we have previously computed its total height in \l_tmpb_dim).

\skip_vertical:n { - \l_tmpb_dim }
}
\right #2
\c_math_toggle_token
\box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
\box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
\box_use_drop:N \l_tmpa_box

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in \g_@@_width_last_col_dim: see p. 49).
\bool_if:NT \g_@@_last_col_found_bool
{ \skip_horizontal:n \g_@@_width_last_col_dim
\skip_horizontal:n \arraycolsep
}
\@@_after_array:
}

This is the end of the environment {NiceArrayWithDelims}.

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).

\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 }

If all the columns must have the same width (if the user has used the option columns-width or the option auto-column-width of the environment {NiceMatrixBlock}), we will add a row in the array to fix the width of the columns and construct the “col” nodes nm-a-col-j (these nodes will be used by the horizontal open dotted lines and by the commands \@@_vdottedline:n). We have written a dedicated function for that job.
\@@_create_row_of_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 }
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

45
The body of the environment, which is stored in the argument \#1, is now split 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:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

We need a global affectation because, when executing \l_tmpa_tl, we will exit the first cell of the array.
\seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
\exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
\seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
\@@_create_row_of_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
{ \seq_gclear_new:N \g_@@_cells_seq
\seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
\seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
\l_tmpa_tl \seq_map_function:NN \g_@@_cells_seq \@@_cell_with_light_syntax:n
}
\cs_new_protected:Npn \@@_line_with_light_syntax:n #1
{ \tl_if_empty:nF { #1 } { \@@_line_with_light_syntax_i:n { #1 } }
}
\cs_new_protected:Npn \@@_cell_with_light_syntax:n #1 { & #1 }

The command \@@_create_row_of_col_nodes: will construct the potential last row. That last row (when it is created) is a false row used to fix the width of the columns (when the array is constructed with an option which specify the width of the columns) and create the col-nodes (that nodes will be used by \vdottedline for instance).
\cs_new:Npn \@@_create_row_of_col_nodes:
{ \bool_if:nT { \l_@@_auto_columns_width_bool || \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim } { \crcr \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & } \omit }

First, we put a "col" node on the left of the first column (of course, we have to do that after the \omit).
\bool_if:nTF
{ \l_@@_auto_columns_width_bool
| | \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim
}
\{ \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & } \omit

We compute in \g_tmpa_dim the common width of the columns. We use a global variable because we are in a cell of an \halign and because we have to use this variable in other cells (of the same row). The affectation of \g_tmpa_dim, like all the affectations, must be done after the \omit of the cell.
\bool_if:nTF
{ \l_@@_auto_columns_width_bool
&& ! \l_@@_block_auto_columns_width_bool

46
\begin{verbatim}
800 \dim_gset:Nn \g_tmpa_dim
801 \{ \g_@@_max_cell_width_dim + 2 \arraycolsep \}
802 \}
803 \dim_gset:Nn \g_tmpa_dim
804 \{ \l_@@_columns_width_dim + 2 \arraycolsep \}
805 \skip_horizontal:N \g_tmpa_dim
806 \tikz [ remember-picture, overlay ]
807 \coordinate [ name = nm - \int_use:N \g_@@_env_int - col - 1 ] ;
808
809 We begin a loop over the columns. The integer \texttt{\g_tmpa_int} will be the number of the current column. This integer is not used to fix the width of the column (since all the columns have the same width equal to \texttt{\l_@@_tmpla_dim}) but for the Tikz nodes.
810 \int_gset:Nn \g_tmpa_int 1
811 \bool_if:nTF \g_@@_last_col_found_bool
812 { \prg_replicate:nn { \g_@@_col_total_int - 3 } }
813 { \prg_replicate:nn { \g_@@_col_total_int - 2 } }
814 {
815 \&
816 \omit
817 \int_gincr:N \g_tmpa_int
818 \skip_horizontal:N \g_tmpa_dim
819 \@@_create_col_node:
820
821 For the last column, we want a special treatment because of the final \texttt{\arraycolsep}.
822 \&
823 \omit
824 \int_gincr:N \g_tmpa_int
825 \skip_horizontal:N \g_tmpa_dim
826 \skip_horizontal:n { - \arraycolsep }
827 \@@_create_col_node:
828 \skip_horizontal:N \arraycolsep
829 \}
830 \}
831 \cs_new_protected:Npn \@@_create_col_node:
832 {
833 \tikz [ remember-picture, overlay ]
834 \coordinate 
835 [ 
836 name = nm - \int_use:N \g_@@_env_int - \int_use:N \g_@@_env_int - col - 1 ] ;
837 }
838
839 Here is the preamble for the “first column” (if the user uses the key \texttt{first-col})
840 \tl_const:Nn \c_@@_preamble_first_col_tl
841 \{ 
842 >
843 \}
844 \@@_begin_of_row:
845
846 The contents of the cell is constructed in the box \texttt{\l_tmpa_box} because we have to compute some dimensions of this box.
847 \hbox_set:Nw \l_tmpa_box
848 \c_math_toggle_token
849 \bool_if:NT \l_@@_small_bool \scriptstyle
84
We insert \texttt{\_\_\_\_\_\_\_code_for_first_col_tl...} but we don’t insert it in the potential “first row” and in the potential “last row”:

\begin{verbatim}
\bool_if:nT
  { \int_compare_p:nNn \c@iRow > 0
    \&\&
    ( \int_compare_p:nNn \_\_\_\_\_\_\_last_row_int < 0
      \|\|
      \int_compare_p:nNn \c@iRow < \_\_\_\_\_\_\_last_row_int
    )
  }
  { \_\_\_\_\_\_\_code_for_first_col_tl }
\end{verbatim}

\begin{verbatim}
< \c_math_toggle_token
\hbox_set_end:
\@@_actualization_for_first_and_last_row:

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

\begin{verbatim}
\dim_gset:Nn \g_@@_width_first_col_dim
  { \dim_max:nn \g_@@_width_first_col_dim
    \box_wd:N \l_tmpa_box }
\end{verbatim}

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

\begin{verbatim}
\hbox_overlap_left:n
  { tikz
    remember-picture ,
    inner-sep = \c_zero_dim ,
    minimum-width = \c_zero_dim ,
    baseline
  }
\node
  [ anchor = base ,
    name = nm -
    \int_use:N \g_@@_env_int -
    \int_use:N \c@iRow -
    0 ,
    alias =
    \str_if_empty:NF \l_@@_name_str
      { \_\_\_\_\_\_\_name_str -
        \int_use:N \c@iRow -
        0
      }
  ] { \box_use:N \l_tmpa_box } ;
\skip_horizontal:n { - 2 \arraycolsep }
\end{verbatim}
Here is the preamble for the “last column” (if the user uses the key \last-col).
\tl_const:Nn \c_@@_preamble_last_col_tl
{ >
  \bool_gset_true:N \g_@@_last_col_found_bool
  \int_gincr:N \c@jCol
  \int_gset:Nn \g_@@_col_total_int
  \{ \int_max:nn \g_@@_col_total_int \c@jCol \}

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:Nn \g_@@_col_total_int
\{ \int_max: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_tmpa_box
\c_math_toggle_token
\bool_if:NT \l_@@_small_bool \scriptstyle

We insert \l_@@_code_for_last_col_tl... but we don’t insert it in the potential “first row” and in the potential “last row”.
\bool_if:nT
  { \int_compare_p:nNn \c@iRow > 0 && \int_compare_p:nNn \l_@@_last_row_int < 0 || \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
  \l_@@_code_for_last_col_tl
}
\l <
{ \c_math_toggle_token
\hbox_set_end:
\@@_actualization_for_first_and_last_row:

We actualise the width of the “last column” because we will use this width after the construction of the array.
\dim_gset:Nn \g_@@_width_last_col_dim
{ \dim_max:nn \g_@@_width_last_col_dim
  \{ \box_wd:N \l_tmpa_box \}
}
\skip_horizontal:n \l_@@_right_margin_dim \l_@@_extra_right_margin_dim
\tikz [\remember-picture, inner-sep = \c_zero_dim, minimum-width = \c_zero_dim, baseline}
The environment \texttt{NiceArray} is constructed upon the environment \texttt{NiceArrayWithDelims} but, in fact, there is a flag \texttt{\_\_NiceArray\_bool}. In \texttt{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_@@\_type_env\_str
\{ \str_gset:Nn \g_@@\_type_env\_str { environment \{ NiceArray \} } \}
\endNiceArrayWithDelims
\end{Verbatim}

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

\begin{Verbatim}
\NewDocumentEnvironment{pNiceArray}{}{
\str_if_empty:NT \g_@@\_type_env\_str
\{ \str_gset:Nn \g_@@\_type_env\_str { environment \{ pNiceArray \} } \}
\endNiceArrayWithDelims
\end{Verbatim}

We create the variants of the environment \texttt{NiceArrayWithDelims}. These variants exist since the version 3.0 of \nicematrix.

\begin{Verbatim}
\NewDocumentEnvironment{bNiceArray}{}{
\str_if_empty:NT \g_@@\_type_env\_str
\{ \str_gset:Nn \g_@@\_type_env\_str { environment \{ bNiceArray \} } \}
\endNiceArrayWithDelims
\end{Verbatim}

\begin{Verbatim}
\NewDocumentEnvironment{BNiceArray}{}{
\str_if_empty:NT \g_@@\_type_env\_str
\{ \str_gset:Nn \g_@@\_type_env\_str { environment \{ BNiceArray \} } \}
\endNiceArrayWithDelims
\end{Verbatim}
The environment \{NiceMatrix\} and its variants

14.6 The environment \{NiceMatrix\} and its variants

How to know whether a cell is “empty”

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

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

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

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

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

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

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

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

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

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

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

```
\cs_new_protected:Nn \@@_after_array:
{ 
\int_compare:nNnTF \c@iRow > 0 
\@@_after_array_i:
{ 
    \@@_error:n { Zero~row } 
    \@@_restore_iRow_jCol:
}
}
```

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

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

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.

```
\int_gset_eq:NN \c@jCol \g_@@_col_total_int
\int_compare:nNnT \l_@@_last_col_found_bool { \int_gdecr:N \c@jCol }
```

We fix also the value of `\c@iRow` and `\g_@@_row_total_int` with the same principle.

```
\int_gset_eq:NN \g_@@_row_total_int \c@iRow
\int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gsub:Nn \c@iRow \c_one_int }
```

In the user has used the option `last-row` without value, we write in the `aux` file the number of that last row for the next run.

```
\bool_if:NT \l_@@_last_row_without_value_bool 
{ 
\io_write:N \mainaux \ExplSyntaxOn 
\io_write:Nx \mainaux 
{ \cs_gset:cpn { @@_last_row_ \int_use:N \g_@@_env_int } 
    { \int_use:N \g_@@_row_total_int } 
}\io_write:Nn \mainaux \ExplSyntaxOff
```

If the environment has a name, we also write a value based on the name because it’s more reliable than a value based on the number of the environment.

```
\str_if_empty:NF \l_@@_name_str 
{ 
\io_write:N \mainaux 
{ \cs_gset:cpn { @@_last_row_ \l_@@_name_str } 
    { \int_use:N \g_@@_row_total_int } 
}\io_write:N \mainaux \ExplSyntaxOff
```

---

We remind that the potential “first column” has the number 0.
By default, the diagonal lines will be parallelized\(^{30}\). There are two types of diagonals lines: the \textbackslash Ddots diagonals and the \textbackslash Iddots diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current \texttt{NiceArray} environment.

\begin{verbatim}
\bool_if:NT \l_@@_parallelize_diags_bool
  \int_zero_new:N \l_@@_ddots_int
  \int_zero_new:N \l_@@_iddots_int
\end{verbatim}

The dimensions \texttt{l_@@_delta_x_one_dim} and \texttt{l_@@_delta_y_one_dim} will contain the \(\Delta x\) and \(\Delta y\) of the first \texttt{Ddots} diagonal. We have to store these values in order to draw the others \texttt{Ddots} diagonals parallel to the first one. Similarly \texttt{l_@@_delta_x_two_dim} and \texttt{l_@@_delta_y_two_dim} are the \(\Delta x\) and \(\Delta y\) of the first \texttt{Iddots} diagonal.

\begin{verbatim}
\dim_zero_new:N \l_@@_delta_x_one_dim
\dim_zero_new:N \l_@@_delta_y_one_dim
\dim_zero_new:N \l_@@_delta_x_two_dim
\dim_zero_new:N \l_@@_delta_y_two_dim
\end{verbatim}

The booleans \texttt{g_@@_medium_nodes_bool} and \texttt{g_@@_large_nodes_bool} may be raised directly in cells of the array (for example in commands \texttt{Block}) but also because the user has used the options \texttt{create-medium-nodes} and \texttt{create-large-nodes} (these options raise the booleans \texttt{l_@@_medium_nodes_bool} and \texttt{l_@@_large_nodes_bool} but theses booleans are converted into the global version \texttt{g_@@_medium_nodes_bool} and \texttt{g_@@_large_nodes_bool} before the creation of the array).

\begin{verbatim}
\bool_if:nTF \g_@@_medium_nodes_bool
  \bool_if:NTF \g_@@_large_nodes_bool
    \@@_create_medium_and_large_nodes:
    \@@_create_medium_nodes:
  \end{verbatim}

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

\begin{verbatim}
\bool_if:NT \l_@@_small_bool
  \dim_set:Nn \l_@@_radius_dim { 0.37 \texttt{pt} }
  \dim_set:Nn \l_@@_inter_dots_dim { 0.25 \texttt{em} }
\end{verbatim}

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

\begin{verbatim}
\tikzset
  \every-picture / .style =
  \overlay ,
  \remember-picture ,
\end{verbatim}

\(^{30}\)It’s possible to use the option \texttt{parallelize-diags} to disable this parallelization.
A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

For a closed extremity, we use the normal node and for an open one, we use the “medium node” or, if it exists, the \textit{w}-node.

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

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

\begin{itemize}
\item the first argument is the row of the cell where the command was issued;
\item the second argument is the column of the cell where the command was issued;
\item the third argument is the \textit{x}-value of the orientation vector of the line;
\item the fourth argument is the \textit{y}-value of the orientation vector of the line;
\end{itemize}

This command computes:

\begin{itemize}
\item \texttt{\_\_\_initial\_i\_int} and \texttt{\_\_\_initial\_j\_int} which are the coordinates of one extremity of the line;
\item \texttt{\_\_\_final\_i\_int} and \texttt{\_\_\_final\_j\_int} which are the coordinates of the other extremity of the line;
\item \texttt{\_\_\_initial\_open\_bool} and \texttt{\_\_\_final\_open\_bool} to indicate whether the extremities are open or not.
\end{itemize}

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

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

\psfrag{A}{	exttt{\_\_\_initial\_i\_int}}
\psfrag{B}{	exttt{\_\_\_initial\_j\_int}}
\psfrag{C}{	exttt{\_\_\_final\_i\_int}}
\psfrag{D}{	exttt{\_\_\_final\_j\_int}}
We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean \l_@@_stop_loop_bool will be used to control these loops.

\bool_set_false:N \l_@@_stop_loop_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
{
\int_add:Nn \l_@@_final_i_int { #3 }
\int_add:Nn \l_@@_final_j_int { #4 }
}\bool_if:NTF \l_@@_final_open_bool
{ We test if we are still in the matrix.
\bool_set_false:N \l_@@_final_open_bool
\int_compare:nNnTF \l_@@_final_i_int > \c@iRow
{ \int_compare:nNnT { #3 } = 1
  { \bool_set_true:N \l_@@_final_open_bool }
}
{ \int_compare:nNnTF \l_@@_final_j_int < 1
  { \int_compare:nNnT { #4 } = -1
    { \bool_set_true:N \l_@@_final_open_bool }
  }
  { \int_compare:nNnTF \l_@@_final_j_int > \c@jCol
    { \int_compare:nNnT { #4 } = 1
      { \bool_set_true:N \l_@@_final_open_bool }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
If we are outside the matrix, we have found the extremity of the dotted line and it’s a open extremity.

{ We do a step backwards because we will draw the dotted line upon the last cell in the matrix (we will use the “medium node” of this cell).
\int_sub:Nn \l_@@_final_i_int { #3 }
\int_sub:Nn \l_@@_final_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool
}

If we are in the matrix, we test 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.

{ \@@_if_not_empty_cell:nnTF \l_@@_final_i_int \l_@@_final_j_int
  { \bool_set_true:N \#1 \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.

{ \cs_set:cpn
  { @@_dotted_
    \int_use:N \l_@@_final_i_int -
    \int_use:N \l_@@_final_j_int
  }
}

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

\cs_if_free:cT
  \pgf@sh@ns@nm - 
  \int_use:N \g_@@_env_int - 
  \int_use:N \l_@@_final_i_int - 
  \int_use:N \l_@@_final_j_int 
\}
\bool_if:NF \l_@@_final_open_bool
  \msg_error:nnx { nicematrix } { Impossible~line }
  \int_use:N \l_@@_final_i_int 
  \bool_set_true:N \l_@@_impossible_line_bool 
\}
\}

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

\bool_set_false:N \l_@@_stop_loop_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
\int_sub:Nn \l_@@_initial_i_int \#3 
\int_sub:Nn \l_@@_initial_j_int \#4 
\bool_set_false:N \l_@@_initial_open_bool 
\int_compare:nNnTF \l_@@_initial_i_int < 1 
  { \bool_set_true:N \l_@@_initial_open_bool }
  { \int_compare:nNnT \l_@@_initial_i_int \#3 = 1 
  { \bool_set_true:N \l_@@_initial_open_bool }
  }
\int_compare:nNnTF \l_@@_initial_j_int < 1 
  { \bool_set_true:N \l_@@_initial_open_bool }
  { \int_compare:nNnT \l_@@_initial_j_int \#4 = 1 
  { \bool_set_true:N \l_@@_initial_open_bool }
  }
\int_compare:nNnT \l_@@_initial_j_int > \c@jCol 
  { \int_compare:nNnT \l_@@_initial_j_int \#4 = \{-1\} 
  { \bool_set_true:N \l_@@_initial_open_bool }
  }
\}
\bool_if:NTF \l_@@_initial_open_bool 
  { \int_add:Nn \l_@@_initial_i_int \#3 
  \int_add:Nn \l_@@_initial_j_int \#4 
  \bool_set_true:N \l_@@_stop_loop_bool }
  { \@@_if_not_empty_cell:nnTF \l_@@_initial_i_int \l_@@_initial_j_int 
  { \bool_set_true:N \l_@@_stop_loop_bool }
  { \cs_set:cpn 
    { \@@_dotted 
    \int_use:N \l_@@_initial_i_int - 
    \int_use:N \l_@@_initial_j_int }
  }

57
We test whether the initial extremity of the dotted line is an implicit cell already dotted (by another dotted line). In this case, we can’t draw the line because we have no Tikz node at the extremity of the arrow (and we can’t use the “medium node” or the “large node” because we should use the normal node since the extremity is not open).

\cs_if_free:cT
{
  pgf@sh@ns@nm -
  \int_use:N \g_@@_env_int -
  \int_use:N \l_@@_initial_i_int -
  \int_use:N \l_@@_initial_j_int
}
{
  \bool_if:NF \l_@@_initial_open_bool
  {
    \msg_error:nnx { nicematrix } { Impossible~line }
    \int_use:N \l_@@_initial_i_int
    \bool_set_true:N \l_@@_impossible_line_bool
  }
}

If we have at least one open extremity, we create the “medium nodes” in the matrix\textsuperscript{31}. We remind that, when used once, the command \@@_create_medium_nodes: becomes no-op in the current TeX group.

\bool_if:nT \l_@@_initial_open_bool \@@_create_medium_nodes:
\bool_if:NT \l_@@_final_open_bool \@@_create_medium_nodes:
}

The command \@@_retrieve_coords:nn retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw\textsuperscript{32}. This command has four implicit arguments which are \l_@@_initial_i_int, \l_@@_initial_j_int, \l_@@_final_i_int and \l_@@_final_j_int. The two arguments of the command \@@_retrieve_coords:nn are the suffix and the anchor that must be used for the two nodes.

The coordinates are stored in \g_@@_x_initial_dim, \g_@@_y_initial_dim, \g_@@_x_final_dim, \g_@@_y_final_dim. These variables are global for technical reasons: we have to do an affectation in an environment \{tikzpicture\}.

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

\textsuperscript{31}We should change this. Indeed, for an open extremity of an horizontal dotted line, we use the \texttt{w}-node, if it exists, and not the “medium node”.

\textsuperscript{32}In fact, with diagonal lines, or vertical lines in columns of type \texttt{L} or \texttt{R}, an adjustment of one of the coordinates may be done.
For the horizontal lines with open extremities, we must take into account the “col” nodes created in the environments which have a fixed width of the columns. The following command will recompute the $x$-value of the extremities in this case (erasing the value computed in `$\@\_retrieve\_coords:nn$`).

```
\cs_new_protected:Nn \@\_adjust\_with\_col\_nodes: 
\begin {tikzpicture} [remember\_picture]
\tikz@parse@node \pgfutil@firstofone
(nm - \int\_use:N \g_@@\_env\_int - col - 0 )
\dim_gset:Nn \g_@@x\_initial\_dim \pgf\x
\end {tikzpicture}
```

The command `$\@\_actually\_draw\_Ldots:$` draws the Ldots line using `$\l_@@\_initial\_i\_int$, $\l_@@\_initial\_j\_int$, $\l_@@\_initial\_open\_bool$, $\l_@@\_final\_i\_int$, $\l_@@\_final\_j\_int$ and $\l_@@\_final\_open\_bool$. We have a dedicated command because if is used also by `$\Hdots$`for.

```
\cs_new_protected:Nn \@\_actually\_draw\_Ldots: 
\begin {tikzpicture} [remember\_picture]
\tikz@parse@node \pgfutil@firstofone
(nm - \int\_use:N \g_@@\_env\_int - col - \int\_use:N \c@jCol )
\dim_gset:Nn \g_@@x\_final\_dim \pgf\x
\end {tikzpicture}
```
If a \( w \)-node exists we use the \( w \)-node for the extremity.

\[
\begin{align*}
\text{\texttt{\backslash cs\_if\_exist:cTF}} & \quad \{ \\
\phantom{=} & \quad \text{\texttt{pgf@sh@ns@mm}} \\
\phantom{=} & \quad \text{\texttt{- \int\_use:N \ g\_@@\_env\_int}} \\
\phantom{=} & \quad \text{\texttt{- \int\_use:N \ l\_@@\_initial\_i\_int}} \\
\phantom{=} & \quad \text{\texttt{- \int\_use:N \ l\_@@\_initial\_j\_int - w}} \\
& \quad \} \\
\phantom{=} & \quad \text{\texttt{- w.base\_west}} \\
\phantom{=} & \quad \text{\texttt{- medium.base\_west}} \\
\} \\
\text{\texttt{\backslash .base\_east}} \\
\} \\
\text{\texttt{\backslash bool\_if:NTF \ l\_@@\_final\_open\_bool}} \\
\} \\
\text{\texttt{\backslash cs\_if\_exist:cTF}} \\
\} \\
\text{\texttt{pgf@sh@ns@mm}} \\
\} \\
\text{\texttt{- \int\_use:N \ g\_@@\_env\_int}} \\
\} \\
\text{\texttt{- \int\_use:N \ l\_@@\_final\_i\_int}} \\
\} \\
\text{\texttt{- \int\_use:N \ l\_@@\_final\_j\_int - w}} \\
\} \\
\text{\texttt{- w.base\_east}} \\
\} \\
\text{\texttt{- medium.base\_east}} \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{\backslash \_\_adjust\_with\_col\_nodes:\}} \\
\text{\texttt{\backslash bool\_if:NT \ l\_@@\_initial\_open\_bool}} \\
\} \\
\text{\texttt{\backslash bool\_if:NT \ l\_@@\_final\_open\_bool}} \\
\} \\
\text{\texttt{\backslash bool\_if:NT \ l\_@@\_final\_open\_bool}} \\
\} \\
\text{\texttt{\backslash bool\_if:NT \ l\_@@\_final\_open\_bool}} \\
\} \\
\text{\texttt{\backslash bool\_if:NT \ l\_@@\_final\_open\_bool}} \\
\}
\end{align*}
\]

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte.

\[
\begin{align*}
\text{\texttt{\backslash dim\_gadd:NN \ g\_@@\_y\_initial\_dim \ { 0.53 \ pt } \}} \\
\text{\texttt{\backslash dim\_gadd:NN \ g\_@@\_y\_final\_dim \ { 0.53 \ pt } \}} \\
\text{\texttt{\backslash \_\_draw\_tikz\_line: \}} \\
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{\backslash cs\_new\_protected:NN \ \_\_draw\_Cdots:nn}} \\
\} \\
\text{\texttt{\backslash cs\_if\_free:cT \ { \_\_ dotted \ #1 - #2 }}} \\
\]
For the vertical dots, we have to distinguish different instances because we want really vertical lines. Be careful: it’s not possible to insert the command \@_retrieve_coords:nn in the arguments T and F of the expl3 commands (why?).

\cs_new_protected:Nn \@@_draw_Vdots:nn
\cs_if_free:cT { @@ _ dotted _ #1 - #2 }
\bool_set_false:N \l@@impossible_line_bool
\@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int 0
\bool_if:NF \l@@impossible_line_bool
\@@_retrieve_coords:xx
\bool_if:NF \l@@initial_open_bool
{ - medium.north-west }
{ .south-west }
\bool_if:NF \l@@final_open_bool
{ - medium.south }
{ .north-west }

The boolean \l_ttmpa_bool indicates whether the column is of type \texttt{L} of \texttt{NiceArray}) or may be considered as if.

\bool_set:Nn \l_ttmpa_bool
\dim_compare_p:nNn \g@@x_initial_dim = \g@@x_final_dim
\@@_retrieve_coords:xx
\bool_if:NF \l@@initial_open_bool
{ - medium.north }
{ .south }
\bool_if:NF \l@@final_open_bool
{ - medium.south }
The boolean \texttt{l\_tmpb\_bool} indicates whether the column is of type \texttt{c} (\texttt{C} of \texttt{NiceArray}) or may be considered as if.

\begin{verbatim}
\bool_set:Nn \l_tmpb_bool
{ \dim_compare_p:nNn \g_@@_x_initial_dim = \g_@@_x_final_dim }
\bool_if:NF \l_tmpb_bool
{ \dim_gset:Nn \g_@@_x_initial_dim
\bool_if:NTF \l_tampa_bool \dim_min:nn \dim_max:nn
{ \g_@@_x_initial_dim \g_@@_x_final_dim }
\dim_gset_eq:NN \g_@@_x_final_dim \g_@@_x_initial_dim
} \@@_draw_tikz_line:
}
\end{verbatim}

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.

\begin{verbatim}
\cs_new_protected:Nn \@@_draw_Ddots:nn
{ \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
{ \bool_set_false:N \l_@@_impossible_line_bool
\@@_find_extremities_of_line:nnnn { #1 } { #2 } \c_one_int \c_one_int
\bool_if:NF \l_@@_parallelize_diags_bool
{ \@@_retrieve_coords:xx
\bool_if:NTF \l_@@_initial_open_bool
{ - medium.north~west }
{ .south~east }
\}
\}
\bool_if:NTF \l_@@_final_open_bool
{ - medium.south~east }
{ .north~west }
}

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

\begin{verbatim}
\bool_if:NT \l_@@_parallelize_diags_bool
{ \int_incr:N \l_@@_ddots_int
\int_compare:nNnTF \l_@@_ddots_int = \c_one_int

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

\end{verbatim}
If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate \texttt{\_g\_y\_initial\_dim}.

\begin{verbatim}
\dim_gset:Nn \g_\_y\_final\_dim
{ \g_\_y\_initial\_dim + \l_\_delta_y\_one\_dim \l_\_delta_x\_one\_dim \dim_ratio:nn \l_\_delta_y\_two\_dim \l_\_delta_x\_two\_dim }
\end{verbatim}

Now, we can draw the dotted line (after a possible change of \texttt{\_g\_y\_initial\_dim}).

\begin{verbatim}
\_draw_tikz_line:
\end{verbatim}

We draw the \texttt{\_idots} diagonals in the same way.

\begin{verbatim}
\cs_new_protected:Nn \_draw_iddots:nn
{ \cs_if_free:cT { \_dotted \_#1 \_#2 }
{ \bool_set_false:N \l_\_impossible_line_bool
\_find_extremities_of_line:nnnn \l_\_initial\_open_bool \l_\_final\_open_bool
\parallelize_diags_bool
\_retrieve_coords:xx
{ \bool_if:NTF \l_\_initial\_open_bool
{ \_medium.north\_east }
{ \_south\_west }
}
{ \bool_if:NTF \l_\_final\_open_bool
{ \_medium.south\_west }
{ \_north\_east }
}
\_draw_tikz_line:
}\}
\end{verbatim}
The command \NiceMatrixLastEnv is not used by the package 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).

1607 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
1608 { \int_use:N \g_@@_env_int }

14.9 The actual instructions for drawing the dotted line with Tikz

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

1609 \cs_new_protected:Nn \@@_draw_tikz_line: { }
1610 \tl_if_eq:NNTF \@@_initial_open_bool \@@_final_open_bool { } { }
1611 \dim_zero_new:N \l_@@_l_dim
1612 \dim_set:Nn \l_@@_l_dim { \fp_to_dim:n { \sqrt { 1 + ( \dim_use:N \g_@@_x_final_dim - \dim_use:N \g_@@_x_initial_dim ) ^ 2 + ( \dim_use:N \g_@@_y_final_dim - \dim_use:N \g_@@_y_initial_dim ) ^ 2 } } }
1613 \dim_compare:nNnF \l_@@_l_dim = \c_zero_dim { }
1614 \int_set:Nn \l_tmpa_int { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim } { }
1615 \bool_if:NTF \l_@@_initial_open_bool { }
1616 \bool_if:NTF \l_@@_final_open_bool { }
1617 \int_set:Nn \l_tma_int { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim } { }
1618 \int_set:Nn \l_tma_int { \dim_ratio:nn \l_@@_l_dim - \l_@@_dotted_lines_margin_dim } \l_@@_inter_dots_dim
1619 \bool_if:NTF \l_@@_final_open_bool { }
1620 \int_set:Nn \l_tma_int { }
1621 \dim_compare:nNnF \l_@@_l_dim = \c_zero_dim { }
1622 \int_set:Nn \l_tma_int { }

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

The integer \l_tma_int is the number of dots of the dotted line.
The dimensions \l@tmpa_dim and \l@tmpb_dim are the coordinates of the vector between two dots in the dotted line.

\begin{verbatim}
\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
\}
\end{verbatim}

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@tmph\_int.

\begin{verbatim}
\int_set:Nn \l@tmph\_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 }
\}
\end{verbatim}

In the loop over the dots (\int_step_inline:nnnn), 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.

\begin{verbatim}
\dim_gadd:Nn \l@x_initial_dim
\{ ( \l@x_final_dim - \l@x_initial_dim ) * \dim_ratio:nn
\l@inter_dots_dim \l@l_dim
\}
\dim_gadd:Nn \l@y_initial_dim
\{ ( \l@y_final_dim - \l@y_initial_dim ) * \dim_ratio:nn
\l@inter_dots_dim \l@l_dim \}
\end{verbatim}

(In a multiplication of a dimension and an integer, the integer must always be put in second position.)
14.10 User commands available in the new environments

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

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

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

\cs_new_protected:Nn \@@_add_to_empty_cells: 
\cs_gset:cpx
\{ \@@_empty \int_use:N \c@iRow - \int_use:N \c@jCol \}
\{ \int_use:N \g_@@_env_int \}

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

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

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

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

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

\NewDocumentCommand \@@_Iddots \s 
\{ \bool_if:nF { #1 } { \@@_instruction_of_type:n { Iddots } } \bool_if:NF \l_@@_nullify_dots_bool { \phantom \@@_iddots } \@@_add_to_empty_cells: \}
The command $\texttt{\@@_Ddots}$ will be linked to $\texttt{\hspace}$ in $\texttt{\{NiceArray\}}$.

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

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

This command must not be protected since it begins with $\texttt{\multicolumn}$.

The command $\texttt{\@@_Hdotsfor_i}$ is defined with the tools of $\texttt{xparse}$ because it has an optionnal argument. Note that such a command defined by $\texttt{\NewDocumentCommand}$ is protected and that's why we have put the $\texttt{\multicolumn}$ before (in the definition of $\texttt{\@@_Hdotsfor:}$).
\@@_draw_Hdotsfor:nnn
\{ \int_use:N \c@iRow \}
\{ \int_use:N \c@jCol \}
\{ #2 \}
\prg_replicate:nn { #2 - 1 } { \multicolumn { 1 } { C } { } }
\}
\cs_new_protected:Nn \@@_draw_Hdotsfor:nnn
\{ \bool_set_false:N \l_@@_initial_open_bool
\bool_set_false:N \l_@@_final_open_bool
\}

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

For the column, it's a bit more complicated.
\int_compare:nNnTF #2 = 1
\{
\int_set:Nn \l_@@_initial_j_int 1
\bool_set_true:N \l_@@_initial_open_bool
\}
\{
\int_set:Nn \l_@@_final_j_int { #2 - 1 }
\@@_if_not_empty_cell:nnTF \l_@@_initial_i_int \l_@@_initial_j_int
\{ \int_set:Nn \l_@@_initial_j_int { #2 - 1 } \}
\{
\int_set:Nn \l_@@_initial_j_int {#2}
\bool_set_true:N \l_@@_initial_open_bool
\}
\}
\int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
\{
\int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
\bool_set_true:N \l_@@_final_open_bool
\}
\{
\int_set:Nn \l_@@_final_j_int { #2 + #3 }
\@@_if_not_empty_cell:nnTF \l_@@_final_i_int \l_@@_final_j_int
\{ \int_set:Nn \l_@@_final_j_int { #2 + #3 } \}
\{
\int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
\bool_set_true:N \l_@@_final_open_bool
\}
\}
\bool_if:nT { \l_@@_initial_open_bool || \l_@@_final_open_bool }
\@@_create_medium_nodes:
\@@_actually_draw_Ldots:

We declare all the cells concerned by the Hdotsfor as “dotted” (for the dotted lines created by \Cdots, \Ldots, etc., this job is done by \@@_find_extremities_of_line:nnnn). This declaration is done by defining a special control sequence (to nil).
\int_step_inline:nnn { #2 } { #2 + #3 - 1 }
\{ \cs_set:cpn { \@@_dotted_#1 -##1 } { } \}

14.11 The command \line accessible in code-after

In the code-after, the command \@@_line:nn will be linked to \line. This command takes two arguments which are the specification of two cells in the array (in the format i-j) and draws a dotted line between these cells.
First, we write a command with an argument of the format \(i-j\) and applies the command \texttt{\inteval:n}\footnote{Indeed, we want that the user may use the command \texttt{\line} in \texttt{code-after} with \LaTeX\ counters in the arguments  — with the command \texttt{\value}.} to \(i\) and \(j\); this must not be protected (and is, of course fully expandable).

With the following construction, the command \texttt{\@@doubleinteval: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{\@@doubleinteval:n} is fully expandable).

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

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

The command \texttt{\hdottedline} draws an horizontal dotted line to separate two rows. Similarly, the letter “:” in the preamble draws a vertical dotted line (the letter can be changed with the option \texttt{\line} in \texttt{code-after} with \LaTeX\ counters in the arguments — with the command \texttt{\value}.}
letter-for-dotted-lines). Both mechanisms write instructions in the code-after. The actual instructions in the code-after use the commands \@_hdottedline:n and \@_vdottedline:n.

We want the horizontal lines at the same position\textsuperscript{34} as the line created by \hline (or \hdashline of arodshln). That’s why we use a \noalign to insert a box with a \dotfill.

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

\begin{verbatim}
\bool_if:NTF \c_@@_draft_bool
{ \cs_set_eq:NN \@@_dotfill: \prg_do_nothing: }
{ \cs_set:Npn \@@_dotfill:
{ \noalign
{ \bool_gset_true:N \g_@@_large_nodes_bool
\cs_if_exist:cTF { @@_width_ \int_use:N \g_@@_env_int }
{ \dim_set_eq:Nc \l_tmpa_dim { @@_width_ \int_use:N \g_@@_env_int } }
{ \dim_set:Nn \l_tmpa_dim { 5 mm } }
\hbox_overlap_right:n
{ \bool_if:nT
{ \l_@@_NiceArray_bool
&& ! \l_@@_exterior_arraycolsep_bool
&& \int_compare_p:nNn \l_@@_first_col_int > 0 }
{ \skip_horizontal:n { - \arraycolsep } }
\hbox_to_wd:nn
{ \l_tmpa_dim + 2 \arraycolsep
- \l_@@_left_margin_dim - \l_@@_right_margin_dim }
\@@_dotfill: }
}
} }
\end{verbatim}

\textsuperscript{34}In fact, almost the same position because of the width of the line: the width of a dotted line is not the same as the width of a line created by \hline.
We should allow the letter “;” in the first position of the preamble but that would need a special programmation.

\int_compare:nNnTF \#1 = 0
\{ \@@_error:n { Use-of:-in-first-position } \}
\{
\bool_if:NF \c_@@_draft_bool
\{
\dim_zero_new:N \g_@@_x_initial_dim
\dim_zero_new:N \g_@@_y_initial_dim
\dim_zero_new:N \g_@@_x_final_dim
\dim_zero_new:N \g_@@_y_final_dim
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_final_open_bool
\}
\cs_if_exist:cTF
\{ pgf@sh@ns@nm -\int_use:N \g_@@_env_int - col - \#1 \}
\{
\begin{tikzpicture} [ remember-picture ]
\tikz@parse@node\pgfutil@firstofone
( col - \#1 )
\dim_gset:Nn \g_@@_x_initial_dim \pgf@x
\dim_gset:Nn \g_@@_x_final_dim \pgf@x
\dim_gset:Nn \g_@@_y_final_dim \pgf@y
\end{tikzpicture}
\dim_gset:Nn \g_@@_y_initial_dim { - \c_max_dim }
\int_step_inline:nn \c@jCol
\{
\begin{tikzpicture} [ remember-picture ]
\tikz@parse@node\pgfutil@firstofone
( 1 - \#1 . north-east )
\dim_gset:Nn \g_@@_y_initial_dim \pgf@y
\end{tikzpicture}
\}
\}
\}
\cs_if_exist:cTF
\{ pgf@sh@ns@nm -\int_use:N \g_@@_env_int - col - \#1 \}
\{
\begin{tikzpicture} [ remember-picture ]
\tikz@parse@node\pgfutil@firstofone
( 1 - \#1 . large .north-east )
\dim_gset:Nn \g_@@_x_initial_dim \pgf@x
\dim_gset:Nn \g_@@_y_initial_dim \pgf@y
\tikz@parse@node\pgfutil@firstofone
( \int_use:N \c@iRow - \#1 - large .south-east )
\dim_gset:Nn \g_@@_x_final_dim \pgf@x
\dim_gset:Nn \g_@@_y_final_dim \pgf@y
\end{tikzpicture}
\}
\}

If not, we use the “large node”.

\cs_if_exist:cTF
\{ pgf@sh@ns@nm -\int_use:N \g_@@_env_int - 1 - \#1 - w \}
\{
\begin{tikzpicture} [ remember-picture ]
\tikz@parse@node\pgfutil@firstofone
( 1 - \#1 - large .north-east )
\dim_gset:Nn \g_@@_x_initial_dim \pgf@x
\dim_gset:Nn \g_@@_y_initial_dim \pgf@y
\tikz@parse@node\pgfutil@firstofone
( \int_use:N \c@iRow - \#1 - large .south-east )
\dim_gset:Nn \g_@@_x_final_dim \pgf@x
\dim_gset:Nn \g_@@_y_final_dim \pgf@y
\end{tikzpicture}
\}

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

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

71
\end{tikzpicture}
\dim_gadd:Nn \g_@@_x_initial_dim \arraycolsep
\dim_gadd:Nn \g_@@_x_final_dim \arraycolsep
}
}
\@@_draw_tikz_line:
}
}
}

14.13 The vertical rules

We don’t want that a vertical rule drawn by the specifier “|” extends in the eventual “first row” and “last row” of the array.

The natural way to do that would be to redefine the specifier “|” with \newcolumntype:

\newcolumntype { | }
{ ! { \int_compare:nNnF \c@iRow = 0 \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 will do a redefinition of the macro \@arrayrule of array and this redefinition will add \@@_vline: instead of \vline to the preamble.

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

\cs_new_protected:Npn \@@_vline:
{
\int_compare:nNnTF \l_@@_first_col_int = 0
{ \int_compare:nNnTF \c@jCol = 0
{ \int_compare:nNnTF \l_@@_first_row_int = 0
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
 }
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }
{ \int_compare:nNnF \c@iRow = 0
{ \int_compare:nNnF \c@iRow = \l_@@_last_row_int
 \@@_vline_i:
 }
 }\end{verbatim}

72
If \texttt{colortbl} is loaded, the following macro will be redefined (in a \texttt{\AtBeginDocument}) to take into account the color fixed by \texttt{\arrayrulecolor} of \texttt{colortbl}.

\begin{verbatim}
\cs_set_eq:NN \@@_vline_i: \vline
\end{verbatim}

We give now the definition of \texttt{\OnlyMainNiceMatrix}. Internally, it is not used by \texttt{nicematrix}. It’s only a facility given to the final user, which may be useful in the definitions of new columns types (with \texttt{\newcolumntype}).

First, we give the definition of \texttt{\OnlyMainNiceMatrix} in the general case: it’s no-op (thus, a definition of column type may be used outside the environments of \texttt{nicematrix}, in \texttt{\{}array\texttt{}}, etc.).

\begin{verbatim}
\cs_set_eq:NN \OnlyMainNiceMatrix \use:n
\end{verbatim}

Now, we give the definition of \texttt{\OnlyMainNiceMatrix} which will be used in the environments of \texttt{nicematrix}. This command \texttt{\@@\OnlyMainNiceMatrix:n} will be linked to \texttt{\OnlyMainNiceMatrix} in \texttt{\@@\pre_array}. This command is “fully expandable” and that’s why we have not protected it, even tough this characteristic will probably not be used.

\begin{verbatim}
\cs_new:Npn \@@_OnlyMainNiceMatrix:n #1
\end{verbatim}

\subsection{The environment \texttt{\{}NiceMatrixBlock\texttt{}}}

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

\begin{verbatim}
\bool_new:N \l_@@_block_auto_columns_width_bool
\keys_define:nn { NiceMatrix / NiceMatrix\Block }
\end{verbatim}

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

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

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).

\begin{verbatim}
\bool_if:NT \l_@@_block_auto_columns_width_bool
  { \iow_now:Nn \@mainaux \ExplSyntaxOn
    \iow_now:Nx \@mainaux
    { \cs_gset:cpn { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
        { \dim_use:N \g_@@_max_cell_width_dim }
    }
    \iow_now:Nn \@mainaux \ExplSyntaxOff
  }
\end{verbatim}

14.15 The extra nodes

First, two variants of the functions \dim_min:nn and \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: \@@_create_medium_nodes:, \@@_create_large_nodes: and \@@_create_medium_and_large_nodes:. They must not be used in the code-after because the code-after is executed in a scope of prefix name of Tikz.

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 \@@_computations_for_medium_nodes: to do these computations.

The command \@@_computations_for_medium_nodes: must be used in a \{tikzpicture\}. For each row \(i\), we compute two dimensions \l_@@_row_i_min_dim and \l_@@_row_i_max_dim. The dimension \l_@@_row_i_min_dim is the minimal \(y\)-value of all the cells of the row \(i\). The dimension \l_@@_row_i_max_dim is the maximal \(y\)-value of all the cells of the row \(i\).

Similarly, for each column \(j\), we compute two dimensions \l_@@_column_j_min_dim and \l_@@_column_j_max_dim. The dimension \l_@@_column_j_min_dim is the minimal \(x\)-value of all the cells of the column \(j\). The dimension \l_@@_column_j_max_dim is the maximal \(x\)-value of all the cells of the column \(j\).

Since these dimensions will be computed as maximum or minimum, we initialize them to \c_max_dim or \(-\c_max_dim\).

\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_eq:cN { \l_@@_row_\@@_i: _max_dim } \c_max_dim
        \dim_zero_new:c { \l_@@_column_\@@_i: _min_dim } \dim_set_eq:cN { \l_@@_column_\@@_i: _min_dim } \c_max_dim
        \dim_zero_new:c { \l_@@_column_\@@_i: _max_dim } \dim_set_eq:cN { \l_@@_column_\@@_i: _max_dim } \c_max_dim
      }
  }
\end{verbatim}

74
We begin the two nested loops over the rows and the columns of the array.

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

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

Here is the command \texttt{\_create_medium_nodes}: When this command is used, the “medium nodes” are created. These nodes won’t be constructed twice because when used once, this command becomes no-op.
The command \@@_create_large_nodes: must be used when we want to create only the “large nodes” and not the medium ones (if we want to create both, we have to use the command \@@_create_medium_and_large_nodes:). However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. That’s why we use first \@@_computations_for_medium_nodes: and then the command \@@_computations_for_large_nodes:.

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.

For “medium nodes”, we use a command \@@_create_nodes: because this command will also be used for the creation of the “large nodes” (after changing the value of name-suffix).

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.
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.

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

We can eventually draw the rectangular node for the cell (`\@@_i-\@@_j`). This node is created with the Tikz library `fit`. Don’t forget that the Tikz option `name suffix` has been set to `-medium` or `-large`.
Now, we create the nodes for the cells of the \textbackslash multicolumn. We recall that we have stored in \texttt{\_\_\_multicolumn\_cells\_seq} the list of the cells where a \texttt{\_\_\_multicolumn} with \( n > 1 \) was issued and in \texttt{\_\_\_multicolumn\_sizes\_seq} the correspondant values of \( n \).

\seq_mapthread_function:NNN \_\_\_multicolumn\_cells\_seq \_\_\_multicolumn\_sizes\_seq \_\_\_node_for_multicolumn:nn

We can now compute the width of the array (used by \texttt{\_\_\_dottedline}). We should modify this point because it’s a waste to construct all the “large nodes” only for computing the width of the array.

\cs_new_protected:Npn \_\_\_compute\_width\_of\_array:
\begin{tikzpicture}[remember picture, overlay]
\tikz@parse@node \pgfutil@firstofone
(nm - \int_use:N \g_@@_env_int - 1 - 1 - large .north~west)
\dim_gset:Nn \g_tmpa_dim \pgf@x
\tikz@parse@node \pgfutil@firstofone
(nm - \int_use:N \g_@@_env_int - 1 - \int_use:N \c@jCol - large .north~east)
\dim_gset:Nn \g_tmpb_dim \pgf@x
\end{tikzpicture}
\iow_now:Nn \@mainaux \ExplSyntaxOn
\iow_now:Nx \@mainaux
{\cs_gset:cpn {@@width_\int_use:N \g_@@_env_int} \dim_eval:n \{ \g_tmpb_dim - \g_tmpa_dim \}}
\iow_now:Nn \@mainaux \ExplSyntaxOff
\cs_new_protected:Npn \_\_\_extract\_coords: #1 - #2 \q_stop
\begin{tikzpicture}
\coordinate (@@south~west) at
(\dim_use:c {l_@@_column_@@_j:min_dim},\dim_use:c {l_@@_row_@@_i:min_dim});
\coordinate (@@north~east) at
(\dim_use:c {l_@@_column_\int_eval:n \{@@_j:+#2-1\}:max_dim},\dim_use:c {l_@@_row_\@@_i:max_dim});
\node[nodecontents = {},fit = (@@south-west)(@@north-east),innersep = \c_zero_dim,name = nm - \int_use:N \g_@@_env_int - \@@_i: - \@@_j:,alias = \str_if_empty:NF \l_@@_name_str}
14.16 Block matrices

The code in this section if for the construction of block matrices. It has no direct link with the environment \{NiceMatrixBlock\}.

The following command will be linked to \Block in the environments of nicematrix. We define it with \NewDocumentCommand of xparse because it has an optional argument between < and > (for TeX instructions put before the math mode of the label)

\NewDocumentCommand \@@_Block: { m D < > { } m } { \@@_Block_i #1 \q_stop { #2 } { #3 } }

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

\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnn { #1 } { #2 } { #3 } { #4 } }

Now, the arguments have been extracted: \(\#1\) is \(i\) (the number of rows of the block), \(\#2\) is \(j\) (the number of columns of the block), \(\#3\) are the tokens to put before the math mode and \(\#4\) is the label of the block. The following command must not be protected because it contains a command \multicolumn (in the case of a block of only one row).

\cs_new:Npn \@@_Block_ii:nnnn #1 #2 #3 #4
\begin{tikzpicture} \[ remember-picture \] \coordinate(nm - \int_use:N \g_@@_env_int - Block - \int_use:N \c@iRow - \int_use:N \c@jCol) \end{tikzpicture}
\bool_gset_true:N \g_@@_medium_nodes_bool

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

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

The following command \@@_Block_iii:nnnnn will be used in the code-after.

\cs_new_protected:Npn \@@_Block_iii:nnnnn \#1 \#2 \#3 \#4 \#5
{ \bool_if:nTF \int_compare:nNnT { \#1 } = 1
{ \begin{tikzpicture} \[ remember-picture \] \coordinate(nm - \int_use:N \g_@@_env_int - Block - \int_use:N \g_@@_env_int - Block - \int_use:N \c@iRow - \int_use:N \c@iRow - \int_use:N \c@jCol) \end{tikzpicture} }\bool_gset_true:N \g_@@_medium_nodes_bool

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

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

The following command \@@_Block_iii:nnnnn will be used in the code-after.

\cs_new_protected:Npn \@@_Block_iii:nnnnn \#1 \#2 \#3 \#4 \#5
{ \bool_if:nTF \int_compare:nNnT { \#1 } = 1
{ \begin{tikzpicture} \[ remember-picture \] \coordinate(nm - \int_use:N \g_@@_env_int - Block - \int_use:N \g_@@_env_int - Block - \int_use:N \c@iRow - \int_use:N \c@iRow - \int_use:N \c@jCol) \end{tikzpicture} }\bool_gset_true:N \g_@@_medium_nodes_bool

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

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

The following command \@@_Block_iii:nnnnn will be used in the code-after.
If the block has only one row, we have to do a special work in order to have the contains of the node aligned with the contents of the other rows of the array.

\int_compare_p:nNn { #4 } > \c@jCol
{ \msg_error:nnnn { nicematrix } { Block-too-large } { #1 } { #2 } }

First, we compute in \l_tmpa_dim the \textit{y}-value of the baseline of the row. We have constructed a special node of shape \texttt{coordinate} in this order.

\begin { tikzpicture }
\tikz@parse@node \pgfutil@firstofone (Block-#1-#2)
\dim_set:Nn \l_tmpa_dim \pgf@y
\node

[ fit = ( #1 - #2 - medium . north-west )
  ( #3 - #4 - medium . south-east ) ,
  inner-sep = 0 pt ,
]
(#1-#2) { } ;

With the following instruction, we retrieve the \textit{x}-value and the \textit{y}-value of the center of the block. We will only use the \textit{x}-value, available in \texttt{pgf@x}.

\begin { tikzpicture }
\tikz@parse@node \pgfutil@firstofone (#1-#2)
\path (\pgf@x,\l_tmpa_dim) node [ anchor = base ] { #5 } ;
\end { tikzpicture }

If the number of rows is different of 1, it’s necessary to create two Tikz nodes because we want the label \texttt{#5} really drawn in the \texttt{center} of the node.

\begin { tikzpicture }
\node

[ fit = ( #1 - #2 - medium . north-west )
  ( #3 - #4 - medium . south-east ) ,
  inner-sep = 0 pt ,
]
\node at (#1-#2.center) { #5 } ;
\end { tikzpicture }

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

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

14.17 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 }
14.18 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 - 1 } { & }
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \}
\prg_replicate:nn { \l_@@_nb_rows_int }
{\int_compare:nNnT \l_@@_first_col_int = 0 { & } \int_compare:nNnT \l_@@_first_col_int = 0 { & } }
\prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \}
\int_compare:nNnT \l_@@_last_row_int > { -2 } {
\int_compare:nNnT \l_@@_first_col_int = 0 { & } \int_compare:nNnT \l_@@_first_col_int = 0 { & } }
\prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
\int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \}
\end { NiceArrayWithDelims }
\cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
{\cs_set_protected:cpn { #1 AutoNiceMatrix }{
\str_gset:Nx \g_@@_type_env_str { command ~ \c_backslash_str #1 AutoNiceMatrix }
\AutoNiceMatrixWithDelims { #2 } { #3 }
}}\@@_define_com:nnn p ( ) \@@_define_com:nnn b [ ] \@@_define_com:nnn v | | \@@_define_com:nnn V \| \| \@@_define_com:nnn B \{ \}

14.19 We process the options

We process the options when the package is loaded (with \usepackage) but we recommend to use \NiceMatrixOptions instead.
We must process these options after the definition of the environment \texttt{NiceMatrix} because the option \texttt{renew-matrix} executes the code \texttt{\cs_set_eq:NN \env@matrix \NiceMatrix}.

Of course, the command \texttt{\NiceMatrix} must be defined before such an instruction is executed.

\begin{verbatim}
\keys_define:nn { NiceMatrix / Package }
\begin{tabular}{@{}rl}
renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
renew-dots .value_forbidden:n = true ,
renew-matrix .code:n = \@@_renew_matrix: ,
renew-matrix .value_forbidden:n = true ,
transparent .meta:n = \{ renew-dots , renew-matrix \},
transparent .value_forbidden:n = true ,
obsolete-environments .code:n =
\end{tabular}
\\end{verbatim}

14.20  Error messages of the package

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

\begin{verbatim}
\@@_msg_new:nn { last-col-non-empty-for-NiceArray }
\begin{tabular}{@{}r}
In-the-\texttt{\g_@@_type_env_str}, you-must-use-the-option-
'last-col'-without-value.\\However,-you-can-go-on-for-this-time-
(the-value-\texttt{\l_keys_value_tl}'-will-be-ignored).
\end{tabular}
\end{verbatim}

\begin{verbatim}
\@@_msg_new:nn { last-col-empty-for-NiceMatrix }
\begin{tabular}{@{}r}
In-the-\texttt{\g_@@_type_env_str}, you-can't-use-the-option-
'last-col'-without-value.-You-must-give-the-number-of-that-last-column.\\If-you-go-on-this-option-will-be-ignored.
\end{tabular}
\end{verbatim}

\begin{verbatim}
\@@_msg_new:nn { Block-too-large }
\begin{tabular}{@{}r}
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.
\end{tabular}
\end{verbatim}

\begin{verbatim}
\@@_msg_new:nn { Impossible-line }
\begin{tabular}{@{}r}
A-dotted-line-can't-be-drawn-because-you-have-not-put-
all-the-ampersands-required-on-the-row-#1.\\If-you-go-on,-this-dotted-line-will-be-ignored.
\end{tabular}
\end{verbatim}

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

\begin{verbatim}
\@@_msg_new:nn { Yet-in-env }
\begin{tabular}{@{}r}
Environments-\texttt{\{NiceArray\}}-(or-\texttt{\{NiceMatrix\}},-etc.)-can't-be-nested.\\end{tabular}
\end{verbatim}
This error is fatal.

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

\@@_msg_new:nn { Option-Transparent-suppressed }
\{ The option 'Transparent' has been renamed 'transparent'.\}
If you go on this command will be ignored.

\@@_msg_new:nn { Option-RenewMatrix-suppressed }
\{ The option 'RenewMatrix' has been renamed 'renew-matrix'.\}
If you go on this command will be ignored.

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

\@@_msg_new:nnn { Unknown-key-for-NiceMatrixOptions }
\{ The key '\tl_use:N\l_keys_key_tl' is unknown for the command \textbackslash token_to_str:N \textbackslash NiceMatrixOptions. \}
If you go on, it will be ignored. \}
For a list of the available keys, type H <return>.

\@@_msg_new:nnn { Unknown-option-for-NiceArray }
\{ The option '\tl_use:N\l_keys_key_tl' is unknown for the environment \textbackslash \{NiceArray\}. \}
If you go on, it will be ignored. \}
For a list of the available options, type H <return>.
The available options are (in alphabetic order): b, c, code-after, code-for-first-col, code-for-first-row, code-for-last-col, code-for-last-row, columns-width, create-extra-nodes, create-medium-nodes, create-large-nodes, dotted-lines-margin, end-of-row, extra-left-margin, extra-right-margin, first-col, first-row, hlines, last-col, last-row, left-margin, light-syntax, name, nullify-dots, parallelize-diags, renew-dots, right-margin, small, and t.

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 t, c and b).

\texttt{\@@_msg_new:nnn \{ Unknown-option-for-NiceMatrix \}

\{ The-option-\texttt{'tl_use:\\l_keys_key_tl'}-is-unknown-for-the- \\
\texttt{\g_@@_type_env_str. \}\} 

If you go on, it will be ignored. \\ For a list of the available options, type H <return>.

\} 

\{ The-available-options-are-(in-alphabetic-order):- 

code-after,- code-for-first-col,- code-for-first-row,- code-for-last-col,- code-for-last-row,- columns-width,- create-extra-nodes,- create-medium-nodes,- create-large-nodes,- dotted-lines-margin,- end-of-row,- extra-left-margin,- extra-right-margin,- first-col,- first-row,- hlines,- last-col,- last-row,- left-margin,-
light-syntax,-
name,-
nullify-dots,-
parallelize-diags,-
renew-dots,-
right-margin,-
and small.
}

\@@_msg_new:nnn { Duplicate-name }
\}
\The-name-’\_keys_value_tl’-is-already-used-and-you-shouldn’t-use-
the-same-environment-name-twice.-You-can-go-on,-but,-
maybe,-you-will-have-incorrect-results-especially-
if-you-use-’columns-width=auto’.-If-you-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~ }.
}

\@@_msg_new:nn { Option-auto-for-columns-width }
\{You-can’t-give-the-value-’auto’-to-the-option-’columns-width’-here.-
If-you-go-on,-the-option-will-be-ignored.
\}

\@@_msg_new:nn { Zero-row }
\{There-is-a-problem.-Maybe-your-’\g_@@_type_env_str’-is-empty.-
Maybe-you-have-used-L,-C-and-R-instead-of-L,-C-and-R-in-the-preamble-
of-your-environment. \}
If-you-go-on,-the-result-may-be-incorrect.
}

\@@_msg_new:nn { Use-of-:in-first-position }
\{You-can’t-use-the-column-specifier-’\l_@@_letter_for_dotted_lines_str’-in-the-
first-position-of-the-preamble-of-’\g_@@_type_env_str’. \}
If-you-go-on,-this-dotted-line-will-be-ignored.
}

14.21 Obsolete environments

\@@_msg_new:nn { Obsolete-environment }
\{The-environment-’\{\@currenvir\}’-is-obsolete.-We-should-use-’#1’-instead.-
However,-you-can-go-on-for-this-time.-
If-you-don’t-want-to-see-this-error-again,-you-should-load-’nicematrix’-
with-the-option-’obsolete-environments’.
\}

\NewDocumentEnvironment { pNiceArrayC } { }
\{\@@_error:nn { Obsolete-environment }
\{the-option-’last-col’ \}
\int_zero:N \l_@@_last_col_int
\pNiceArray
\}
\endpNiceArray
\NewDocumentEnvironment { bNiceArrayC } { }
\{\@@_error:nn { Obsolete-environment }
\{the-option-’last-col’ \}
\int_zero:N \l_@@_last_col_int

85
\bNiceArray
}{ \end\bNiceArray }
\NewDocumentEnvironment { BNiceArrayC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-option-'last-col' }
\int_zero:N \l_@@_last_col_int
\BNiceArray }
{ \endBNiceArray }
\NewDocumentEnvironment { vNiceArrayC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-option-'last-col' }
\int_zero:N \l_@@_last_col_int
\vNiceArray }
{ \endvNiceArray }
\NewDocumentEnvironment { VNiceArrayC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-option-'last-col'~and~'first-row' }
\int_zero:N \l_@@_last_col_int
\int_zero:N \l_@@_first_row_int
\VNiceArray }
{ \endVNiceArray }
\NewDocumentEnvironment { pNiceArrayRC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-options-'last-col'~and~'first-row' }
\int_zero:N \l_@@_last_col_int
\int_zero:N \l_@@_first_row_int
\pNiceArray }
{ \endpNiceArray }
\NewDocumentEnvironment { bNiceArrayRC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-options-'last-col'~and~'first-row' }
\int_zero:N \l_@@_last_col_int
\int_zero:N \l_@@_first_row_int
\bNiceArray }
{ \endbNiceArray }
\NewDocumentEnvironment { BNiceArrayRC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-options-'last-col'~and~'first-row' }
\int_zero:N \l_@@_last_col_int
\int_zero:N \l_@@_first_row_int
\BNiceArray }
{ \endBNiceArray }
\NewDocumentEnvironment { vNiceArrayRC } { }
{ \@@_error:nn { Obsolete-environment }
{ the-options-'last-col'~and~'first-row' }
\int_zero:N \l_@@_last_col_int
\int_zero:N \l_@@_first_row_int
\vNiceArray }
{ \endvNiceArray }
15 History

Changes between versions 1.0 and 1.1
The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency). Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2
New environment \{NiceArray\} with column types \(L\), \(C\) and \(R\).

Changes between version 1.2 and 1.3
New environment \{pNiceArrayC\} and its variants. Correction of a bug in the definition of \{BNiceMatrix\}, \{vNiceMatrix\} and \{VNiceMatrix\} (in fact, it was a typo). Options are now available locally in \{pNiceMatrix\} and its variants. The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4
The column types \(w\) and \(W\) can now be used in the environments \{NiceArray\}, \{pNiceArrayC\} and its variants with the same meaning as in the package \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 nicematrix were focused on the continuous dotted lines whereas the version 2.0 of nicematrix provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

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

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option \texttt{code-for-first-row} is now available in the command \texttt{\NiceMatrixOptions}.
Following a discussion on TeX StackExchange\textsuperscript{35}, Tikz externalization is now deactivated in the environments of the extension nicematrix.\textsuperscript{36}

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 \\
  0 & \cdots & 0 \\
  a & \cdots & \cdots \\
  0 & \cdots & 0 
\end{pmatrix} \left( L_i \right)

Changes between version 2.1.3 and 2.1.4

Replacement of some options \texttt{0 \{} \texttt{\}} in commands and environments defined with \texttt{xparse} by \texttt{! 0 \{} \texttt{\}} (because a recent version of \texttt{xparse} introduced the specifier \texttt{!} and modified the default behaviour of the last optional arguments).
See \url{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 \texttt{allow-duplicate-names}.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command \texttt{\hdottedline} (similar to the classical command \texttt{\hline} and the command \texttt{\hdashline} of arydshln).
Possibility to draw vertical dotted lines to separate columns with the specifier \texttt{::} in the preamble (similar to the classical specifier \texttt{\mid} and the specifier \texttt{::} of arydshln).

\textsuperscript{35}Cf. \url{tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package}
\textsuperscript{36}Before this version, there was an error when using nicematrix with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by nicematrix because they use the options \texttt{overlay} and \texttt{remember picture}.
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 (except the dotted lines drawn by \cdottedline,
the symbol : (in the preamble of the array) and \line in code-after.
The starred versions of \Cdots, \Ldots, etc. are now deprecated because, with the new implementation,
they become pointless. These starred versions are no longer documented.
The vertical rules in the matrices (drawn by |) are now compatible with the color fixed by colortbl.
Correction of a bug: it was not possible to use the colon : in the preamble of an array when pdflatex
was used with french-babel (because french-babel activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option small.

Changes between version 3.2 and 3.3

The options first-row, last-row, first-col and last-col are now available in the environments
\NiceMatrix, \pNiceMatrix, \bNiceMatrix, etc.
The option columns-width=auto doesn’t need any more a second compilation.
The options renew-dots, renew-matrix and transparent are now available as package options (as
said in the documentation).
The previous version of nicematrix was incompatible with a recent version of expl3 (released
2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange\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 \texttt{iRow} and \texttt{jCol} available in the cells of the array.
Addition of \texttt{normalbaselines} before the construction of the array: in environments like \texttt{(align)} of \texttt{amsmath} the value of \texttt{\baselineskip} is changed and if the options \texttt{first-row} and \texttt{last-row} were used in an environment of \texttt{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 \texttt{Duplicate-name} in the environments of \texttt{amsmath}.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: \texttt{code-for-first-col}, \texttt{code-for-last-col}, \texttt{code-for-first-row} and \texttt{code-for-last-row}.
New command \texttt{\pAutoNiceMatrix} and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programmation for the command \texttt{\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 \texttt{\multicolumn}.
An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands \texttt{\NiceMatrixLastEnv} and \texttt{\OnlyMainNiceMatrix}.
New options \texttt{create-medium-nodes} and \texttt{create-large-nodes}.

Changes between version 3.9 and 3.10

New option \texttt{light-syntax} (and \texttt{end-of-row}).
New option \texttt{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}.

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{\_Hspace}:                      | 505, 1756 |
|                                                                        | \texttt{\_Idots}:                      | 505, 516, 1750 |
|                                                                        | \texttt{\_Iddots_lines_tl}:            | 559, 1175 |
|                                                                        | \texttt{\_Ldots}:                      | 499, 512, 517, 1726 |
|                                                                        | \texttt{\_Ldots_lines_tl}:             | 556, 1177 |
|                                                                        | \texttt{\l_\_NiceArray Bool}           |          |
|                                                                        | \texttt{\g_\_NiceMatrixBlock int}:     |          |
|                                                                        | \texttt{\g_\_NiceMatrixBlock_int}:     |          |
|                                                                        | \texttt{\l_\_NiceMatrixLast Env}:      |          |
|                                                                        | \texttt{\OnlyMainNiceMatrix: n}:       | 509, 2042 |
|                                                                        | \texttt{\Vdots}:                       | 501, 514, 1738 |
|                                                                        | \texttt{\Vdots_lines_tl}:              | 557, 1173 |
|                                                                        | \texttt{\_actualization_for_first_and_last_row}: |          |
|                                                                        | \texttt{\_actually_draw_Ldots}:         | 319, 344, 871, 942 |

90
\begin{verbatim}
\l_tmpa_dim \dim_eval:n \l_tmpb_dim \dim_gzero_new:N \dim_gzero:N \dim_gzero_new:N \dim_min:n \dim_max:nn \dim_set:Nn \dim_set_eq:NN \dim_sub:NN \dim_use:N \dim_zero:N \dim_zero_new:N \c_max_dim \l_g_tmpa_dim \l_g_tmpb_dim \l_g_tmpc_dim \dim_gzero_new:N \dim_gzero:N \dim_gzero_new:N \dim_min:n \dim_max:nn \dim_set:Nn \dim_set_eq:NN \dim_sub:NN \dim_use:N \dim_zero:N \dim_zero_new:N \c_max_dim \l_g_tmpa_dim \l_g_tmpb_dim \l_g_tmpc_dim
\end{verbatim}
Contents

1 Presentation ................................. 1
2 The environments of this extension 2
3 The continuous dotted lines 2
  3.1 The option nullify-dots 3
  3.2 The command \hdotsfor 3
  3.3 How to generate the continuous dotted lines transparently 4
  3.4 Fine tuning of the dotted lines 5
4 The Tikz nodes created by \texttt{nicematrix} 5
5 The code-after 7
6 The environment \{NiceArray\} 7
7 The exterior rows and columns 8
8 The dotted lines to separate rows or columns 9
9 The width of the columns 10
10 Block matrices 11