The package nicematrix*

F. Pantigny
fpantigny@wanadoo.fr

February 26, 2018

Abstract

The LaTeX package nicematrix gives an environment `{NiceMatrix}` which is similar to the environment `{matrix}` but gives the possibility to draw ellipis dots between the cells of the matrix.

1 Presentation

This package can be used with `xelatex`, `lualatex`, `pdflatex` but also by the classical workflow `latex-dvips-ps2pdf` (or Adobe Distiller). Two compilations may be necessary. This package requires the packages expl3, l3keys2e, xparse, array, mathtools and tikz.

The package nicematrix aims to draw beautiful matrices in a way almost transparent for the user.

Consider, for exemple, the matrix $A = \begin{pmatrix} 1 & \cdots & \cdots & 1 \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix}$

Usually, when using LaTeX and amsmath (or mathtools), such a matrix is composed with an environment `{pmatrix}` and the following code:

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

If we load the package nicematrix with the option `Transparent`, the same code will give the following result:

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

The dotted lines are drawn with Tikz. Two compilations may be necessary.

*This document corresponds to the version 1.1 of nicematrix, at the date of 2018/02/26.

2 How to use nicematrix for new code

2.1 The environment NiceMatrix and its variants

The package nicematrix gives six new environments {NiceMatrix}, {pNiceMatrix}, {bNiceMatrix}, {BNiceMatrix}, {vNiceMatrix} and {VNiceMatrix}. By default, these environments behave almost exactly as the corresponding environments of amsmath (and mathtools): {matrix}, {pmatrix}, {bmatrix}, {Bmatrix}, {vmatrix} and {Vmatrix}.

Inside these environments, five new commands are defined: \Ldots, \Cdots, \Vdots, \Ddots and \Iddots. These commands are intended to be used in place of \dots, \cdots, \vdots, \ddots and \iddots.[1]

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

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

$$\begin{bmatrix} a_1 & \cdots & & a_1 \\ \vdots & & & \\ & a_2 & \cdots & a_2 \\ \vdots & \vdots & \ddots & \\ a_1 & a_2 & & a_n \end{bmatrix}$$

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

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

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

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

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

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

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

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

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

[1] The command \iddots, defined in nicematrix, is a variant of \ddots with dots going forward: $\iddots$. If the command \iddots is already defined (for example if mathdots is loaded), the previous definition is not overwritten. Note that the package yhmath provides a command \adots similar to \iddots.

[2] The precise definition of a "non-empty cell" is given below.

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

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

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

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

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

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

2.2 The option NullifyDots

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

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

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

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

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

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

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

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

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

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

```
\NiceMatrixOptions{NullifyDots}
$D = \begin{pNiceMatrix}
a_0 & b      \\
a_1 & \Vdots \\
a_2 &        \\
a_3 &        \\
a_4 &        \\
a_5 & b
\end{pNiceMatrix}$
```

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

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

3 How to use nicematrix for existing code

The package nicematrix provides an option called `Transparent` for using existing code transparently (this option — and the others — can be set with the `\usepackage` command but nicematrix provides also a dedicated command called `\NiceMatrixOptions`).
In fact, this option is an alias for the conjonction of two options : `RenewDots` and `RenewMatrix`.

- The option `RenewDots`

  With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`[3] are redefined within the environments `{NiceMatrix}` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots`; the command `\dots` ("automatic dots" of `amsmath` — and `mathtools`) is also redefined to behave like `\Ldots`.

- The option `RenewMatrix`

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

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

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

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

4 Technical remarks

4.1 Diagonal lines

By default, all the diagonal lines of a same matrix 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 matrix can have a marked effect on the final result.

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

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

Example with parallelization (default):

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

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

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

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

It's possible to turn off the parallelization with the option `ParallelizeDiagonals` set to `false`:

The same example without parallelization:
`\NiceMatrixOptions{ParallelizeDiagonals=false}`.

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

4.2 The "empty" cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides[4]. However, a empty cell is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands `&`). Indeed, a cell with contents `\hspace*{1cm}` may be considered as empty.

For nicematrix, the precise rules are as follow.

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

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

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

- Each cell whose TeX ouput has a width less than 0.5 pt is empty.

- A cell which contains a command `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` or `\Iddots` and their starred versions is empty. We recall that theses commands should be used alone in a cell.

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

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

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

5 Examples

A tridiagonal matrix :

```
\NiceMatrixOptions{NullifyDots}
$\begin{pNiceMatrix}
a      & b      & 0      &        & \Cdots & 0      \\
b      & a      & b      & \Ddots &        & \Vdots \\
0      & b      & a      & \Ddots &        &        \\
       & \Ddots & \Ddots & \Ddots &        & 0      \\
\Vdots &        &        &        &        & b      \\
0      & \Cdots &        & 0      & b      & a
\end{pNiceMatrix}$
```

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

A permutation matrix :

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

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

An example with `\Iddots`:

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

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

6 Implementation

6.1 Declaration of the package and extensions loaded

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

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

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

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

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

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

```
\RequirePackage{xparse}
```

6.2 Technical definitions

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

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

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

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

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

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

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

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

6.3 The options

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

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

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

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

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

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

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

```
\keys_define:nn {NiceMatrix}
     {ParallelizeDiagonals .bool_set:N = \l_@@_parallelize_diags_bool,
      ParallelizeDiagonals .default:n  = true,
```

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

```
      RenewDots            .bool_set:N = \l_@@_renew_dots_bool,
      RenewDots            .default:n  = true,
```

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

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

Without the option `NullifyDots`, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.). This option is set by default.

```
      NullifyDots          .bool_set:N = \l_@@_nullify_dots_bool ,
      NullifyDots          .default:n  = true,
```

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

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

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

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

6.4 The main functions

The pseudo-environment `\@@_Cell:`–`\@@_end_Cell:` is used to format the cells of the array (except the cells of the first column: see below).

```
\cs_new_protected:Nn \@@_Cell:
   {
```

We increment `\g_@@_column_int`, which is the counter of the columns. We need a global affectation because this command will be used in the cell of a `\halign` (via an environment `{array}`).

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

At the end of the array, the counter `\g_@@_nb_column_int` will contain the total number of columns of the array (even if all the lines don't have the same number of ampersands).

```
    \int_gset:Nn \g_@@_nb_column_int {\int_max:nn \g_@@_nb_column_int \g_@@_column_int}
```

We create a Tikz node for the current cell of the array.

```
    \tikz[remember~picture, inner~sep = 0pt, minimum~width = 0pt, baseline]
       \node [anchor=base] (nm-\int_use:N \g_@@_env_int-
                                \int_use:N \g_@@_line_int-
                                \int_use:N \g_@@_column_int)
       \bgroup $} % $
\cs_new_protected:Nn \@@_end_Cell:
   {$\egroup ;} % $
```

The pseudo-environment `\@@_Cell_First_Column:`–`\@@_end_Cell:` is used to format the cells of the first column of the array. For such a column, we have to increment the counter of the lines (`\g_@@_line_int`) and to initialize the counter of the columns (`\g_@@_column_int`).

```
\cs_new_protected:Nn \@@_Cell_First_Column:
   {\int_gincr:N \g_@@_line_int
    \int_gset:Nn \g_@@_column_int 0
    \@@_Cell:}
```

The environment `{NiceMatrix}` is the main environment of the package `nicematrix`. This environment creates an array similar to the array created by the environment `{matrix}` of amsmath but with Tikz nodes for each cell of the matrix.

```
\NewDocumentEnvironment {NiceMatrix} {}
   {
```

We use a `\aftergroup` to execute `\@@_draw_lines` at the end of the environment (all the LaTeX environments are TeX groups). With this technic, the second part of the environment is the same that the second part of the environment `{matrix}` of amsmath. Therefore, it's easier to redefine the environment `{matrix}` (when the option `RenewMatrix` is used).

```
    \aftergroup \@@_draw_lines:
```

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

```
    \cs_set_eq:NN \Ldots \@@_Ldots
    \cs_set_eq:NN \Cdots \@@_Cdots
    \cs_set_eq:NN \Vdots \@@_Vdots
    \cs_set_eq:NN \Ddots \@@_Ddots
    \cs_set_eq:NN \Iddots \@@_Iddots
    \cs_set_eq:NN \Hspace \@@_Hspace:
    \cs_set_eq:NN \NiceMatrixEndPoint \@@_NiceMatrixEndPoint:
    \bool_if:NF \l_@@_renew_matrix_bool
        {\cs_set_eq:NN \multicolumn \@@_multicolumn:nn}
```

If the option `RenewDots` is used, we redefine the commands `\ldots`, `\cdots`, etc.

```
    \bool_if:NT \l_@@_renew_dots_bool
       {\cs_set_eq:NN \ldots \@@_Ldots
        \cs_set_eq:NN \cdots \@@_Cdots
        \cs_set_eq:NN \vdots \@@_Vdots
        \cs_set_eq:NN \ddots \@@_Ddots
        \cs_set_eq:NN \iddots \@@_Iddots
        \cs_set_eq:NN \dots  \@@_Ldots}
```

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

```
    \int_gincr:N \g_@@_env_int
```

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

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

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

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

The counter \g_@@_line_int will be used to count the lines of the array. At the end of the environment {array}, this counter will give the total number of lines of the matrix.

```
\int_gzero_new:N \g_@@_line_int
```

The counter \g_@@_column_int will be used to count the colums of the array. Since we want to known the total number of columns of the matrix, we also create a counter \g_@@_nb_column_int. These counters are updated in the command \@@_Cell: executed at the beginning of each cell.

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

The next two lines are the same as in the command \env@matrix of amsmath on which all the matrix constructions are built.

```
\hskip -\arraycolsep
\cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

Eventually, the environment {NiceMatrix} is defined upon the environment {array}. We maintain the signification of the counter \c@MaxMatrixCols of amsmath.

```
\int_set:Nn \l_tmpa_int {\c@MaxMatrixCols - 1}
\array{>{\@@_Cell_First_Column:}c<{\@@_end_Cell:}
                *\l_tmpa_int{>{\@@_Cell:}c<{\@@_end_Cell:}}}}
```

The second part of the environment {NiceMatrix} is the same as the second part of the environment {matrix} of amsmath. However, at the end of the environment, the instruction \@@_draw_lines: will be executed because we have put a "\aftergroup \@@_draw_lines:" in the beginning of the environment (therefore, it's possible to implement the option RenewMatrix with \cs_set_eq:NN \env@matrix \NiceMatrix).

```
   {\endarray
    \hskip -\arraycolsep}
```

We create the variants of the environment {NiceMatrix}.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.5 We draw the lines in the matrix

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

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

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

The following variables will be used further.

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

By befault, the diagonal lines will be parallelized[5]. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count the both types in order to known wether a diagonal is the first of its type in the current `{NiceMatrix}` environment.

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

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

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

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

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

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

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

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

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

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

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

[5] It's possible to use the option `ParallelizeDiagonals` to disable this parallelization.

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

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

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

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

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

We test if we are still in the matrix.

```
    \bool_if:nTF { \int_compare_p:nNn \l_@@_final_i_int < 1
                || \int_compare_p:nNn \l_@@_final_i_int > \g_@@_line_int
                || \int_compare_p:nNn \l_@@_final_j_int < 1
                || \int_compare_p:nNn \l_@@_final_j_int > \g_@@_nb_column_int}
```

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

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

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

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

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

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

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

```
\bool_set_eq:NN \l_@@_stop_loop_bool \l_@@_impossible_instruction_bool
\bool_do_until:Nn \l_@@_stop_loop_bool
   {\int_sub:Nn \l_@@_initial_i_int \l_@@_di_int
    \int_sub:Nn \l_@@_initial_j_int \l_@@_dj_int
    \bool_if:nTF
            {   \int_compare_p:nNn \l_@@_initial_i_int < 1
             || \int_compare_p:nNn \l_@@_initial_i_int > \g_@@_line_int
             || \int_compare_p:nNn \l_@@_initial_j_int < 1
```

```
                        || \int_compare_p:nNn \l_@@_initial_j_int > \g_@@_nb_column_int}
                      {\bool_set_true:N \l_@@_impossible_instruction_bool
                       \bool_set_true:N \l_@@_stop_loop_bool}
                      {\@@_if_not_empty_cell:nnT \l_@@_initial_i_int \l_@@_initial_j_int
                            {\bool_set_true:N \l_@@_stop_loop_bool}}
              }
```

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

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

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

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

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

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

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

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

We draw the line.

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

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

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

The command `\@@_retrieve_coords:nn` retrieves the Tikz coordinates of the two extremities of the dotted line we will have to draw [6]. This command has four implicit arguments which are `\l_@@_initial_i_int`, `\l_@@_initial_j_int`, `\l_@@_final_i_int` and `\l_@@_final_j_int`.
The two arguments of the command `\@@_retrieve_coords:nn` are the anchors that must be used for the two nodes.
The coordinates are stored in `\g_@@_x_initial_dim`, `\g_@@_y_initial_dim`, `\g_@@_x_final_dim`, `\g_@@_y_final_dim`. These variables are global for technical reasons: we have to do an affectation in an environment `{pgfpicture}`.

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

[6] In fact, with diagonals lines, a adjustment of one of the coordinates may be done.

```
      \tikz@parse@node\pgfutil@firstofone
                  (nm-\int_use:N \g_@@_env_int-
                      \int_use:N \l_@@_final_i_int-
                      \int_use:N \l_@@_final_j_int.#2)
      \dim_gset:Nn \g_@@_x_final_dim \pgf@x
      \dim_gset:Nn \g_@@_y_final_dim \pgf@y
      \end{tikzpicture} }


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


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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.6 The actual instructions for drawing the dotted line with Tikz

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

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

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

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

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

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

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

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

In the loop over the dots (`\int_step_inline:nnnn`), the dimensions `\g_@@_x_initial_dim` and `\g_@@_y_initial_dim` will be used for the coordinates of the dots.

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

6.7 User commands available in environments NiceMatrix

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

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

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

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

The commands \@@_Ldots, \@@_Cdots, \@@_Vdots, \@@_Ddots and \@@_Iddots will be linked to \Ldots, \Cdots, \Vdots, \Ddots and \Iddots in the environment {NiceMatrix}.

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


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


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


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


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

The command \@@_Hspace: will be linked to \hspace in the environment {NiceMatrix}.

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

The command \@@_NiceMatrixEndPoint: will be linked to \NiceMatrixEndPoint in the environment {NiceMatrix}.

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

6.8 We process the options

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

```
\ProcessKeysOptions {NiceMatrix}
```

6.9 The error messages

```
\msg_new:nnnn {nicematrix}
              {Impossible~instruction}
              {It's~not~possible~to~execute~the~instruction~
               \int_case:nn \l_@@_type_int
                 {0 {\token_to_str:N \Ldots}
                  1 {\token_to_str:N \Cdots}
                  2 {\token_to_str:N \Vdots}
                  3 {\token_to_str:N \Ddots}}~in~the~line~\int_use:N\l_@@_line_int\
               ~and~the~column~\int_use:N\l_@@_column_int\space of~the~matrix~
               because~it's~impossible~to~find~one~of~its~extremities~
               (both~extremities~must~be~non~empty~cells~of~the~matrix).~
               If~you~go~on,~the~instruction~will~be~ignored.}
              {You~can~specify~a~end~of~line~on~a~empty~cell~
               with~\token_to_str:N \NiceMatrixEndPoint.}
\msg_new:nnn {nicematrix}
             {multicolumn~forbidden}
             {The~command~\token_to_str:N \multicolumn\
              is~forbidden~in~the~environment~\{NiceMatrix\}~
              and~its~variants.~The~command~\token_to_str:N \hdotsfor\
              of~amsmath~is~also~forbidden~since~it~uses~
              \token_to_str:N \multicolumn.~You~can~go~on~but~your~line~will~
              probably~be~wrong.}
```

7 History

7.1 Changes between versions 1.0 and 1.1

Option `Silent` (with this option, the impossible instructions are discarded silently).
The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.