The DuckBoat — Beginners' Pond: No more table nightmares with `tabularray`!

Herr Professor Paulinho van Duck

Abstract

In this installment, Prof. van Duck will introduce you to `tabularray`, a package for typesetting tabulars and arrays with LaTeX3.

1 Reputation record!

Hi, (LA)TEX friends!

A sensational event happened on August 3rd, 2022. Prof. Enrico Gregorio, a.k.a. egreg, reached one million reputation points on TEX.SE, quack! He is the first TEX user to smash this record.

You can understand the exceptionality of this fact since on the leading site, Stack Overflow, fewer than ten people have passed that threshold.

I am pleased to have had the occasion to congratulate him personally. We met last summer at the seaside and had a pizza (without pineapple) to celebrate.

In the same period, the funny Ti*k*Zpingus package [3] was created. It is a sort of Ti*k*Zlings spin-off and allows you to draw nice penguins with a very wide set of features.

The picture above is an example of its use.

Now let us move to our current topic. Are you struggling trying to set the height of your table rows? Do you think aligning some cells at the top and some others at the bottom is a mission impossible? Are you bored with typing `\hline` at every row end? Are you going crazy vertically centering a text in a `\multirow`? Would you like to use colors and `booktabs` together?

Do you dream of a package that makes all these amenities with simple options? Sometimes dreams come true, quack!

I am pleased to introduce you to `tabularray`, a recent package for typesetting tabulars and similar environments with many handy options.

Of course, I will not explain all the features of the package, they are a *large* number! Please refer to the package documentation [2] for further details.

Last but not least, let me thank Jianrui Lyu, the brilliant author of the package, for his very accurate review of this article. He is also a user of TopAnswers TEX (`topanswers.xyz/tex`). If you ask a question there, you may get an answer directly from him.

I am also grateful to samcarter and egreg for their suggestions. samcarter also provided the example in Box 13.

Of course, any errors that remain are my sole responsibility.

2 Quack Guide n. 8: The `tabularray` package

As usual, the first thing to do is to load this awesome package in your preamble with
`\usepackage{tabularray}`

Its main environment is `tblr`; it works in both text and math mode. This is already a nice feature, is it not? You can also have a text table in a math environment by setting `mode=text` (more in Section 2.8).

The environment `tblr` has a mandatory argument, where you can specify all the options you like. For example, the standard `l` (left), `c` (center), and `r` (right) can be used for the horizontal alignment; there is also `j` for justified text.

It also has an optional argument, where, for instance, you can choose the baseline of the table. It will not be treated here for reasons of brevity; please refer to the manual [2].

In some of the examples shown in the article, vertical rules are used, but only to better show some `tabularray` features; remember: ***avoid vertical rules in professional tables***, they are generally useless and inelegant, quack!

2.1 Row and column separation (and some other options)

Box 1 shows the difference in default row separation between an ordinary `tabular`/`array` (on the left in the box) and a `tblr` (on the right).

The tables created by `tblr` look better because they have some extra space above and below the rows. You no longer have to fight against `\arraystretch` or `\extrarowheight`, quack!

The size of the vertical space above/below the row (or both) is fully customizable, respectively setting `abovesep`, `belowsep` or `rowsep` as options in `rows=`⟨*styles*⟩.

There are also the corresponding options for `columns` (`leftsep`, `rightsep`, `colsep`, with the obvious meanings).

You can even set the spacing (or any other option) for one or a group of rows/columns only.

With `row{`⟨*number*⟩`}={`⟨*styles*⟩`}` you can set options valid only for the rows indicated by ⟨*number*⟩.

Herr Professor Paulinho van Duck

doi.org/10.47397/tb/44-1/tb136duck-tabularray

Box 1 – `tabular` and `array` vs. `tblr`

```
\begin{tabular}{lcr}
  \hline
  p & v & d \\\hline
  p & dl & q \\\hline
\end{tabular} \hspace{1em} vs.\ \hspace{1em}
\begin{tblr}{lcr}
  \hline
  p & v & d \\\hline
  p & dl & q \\\hline
\end{tblr}\par\vspace{1ex}
$\begin{array}{cc}
  \dfrac{1}{2} & \dfrac{3}{4}  \\
  \dfrac{5}{7} & -\dfrac{9}{10} \\
\end{array}$ \hspace{1em} vs.\ \hspace{1em}
$\begin{tblr}{cc}
  \dfrac{1}{2} & \dfrac{3}{4}  \\
  \dfrac{5}{7} & -\dfrac{9}{10} \\
\end{tblr}$
```

p	v	d
p	dl	q

vs.

p	v	d
p	dl	q

$$\begin{array}{cc} \frac{1}{2} & \frac{3}{4} \\ \frac{5}{7} & -\frac{9}{10} \end{array}$$

vs.

$$\begin{array}{cc} \frac{1}{2} & \frac{3}{4} \\ \frac{5}{7} & -\frac{9}{10} \end{array}$$

This could be a single number, a range ⟨*n*–*m*⟩, a list ⟨*n*,*m*,*p*,*q*⟩, or `even` or `odd` if you need some customizations for even or odd rows.

`odd[`⟨*n*–*m*⟩`]` and `even[`⟨*n*–*m*⟩`]` also accept an optional argument which specifies from/to which row you would like the ⟨*styles*⟩ to apply. If the end row is the last of the table, it can be omitted.

There is also the key for specific columns: `column{`⟨*number*⟩`}={`⟨*styles*⟩`}`

You can also mix `rows`/`columns` for general settings and `row`/`column` for specific ones.

Please note that `tabularray` generally ignores spaces around and within its arguments. For instance,

```
column{3-Z}={yellow!10}
```

is the same as

```
column {  3-Z } = { yellow!10 };
```

only the range cannot be separated by spaces.

Another wonderful feature is the possibility to use `U`, `V`, `W`, `X`, `Y`, and `Z` as row/column numbers, to indicate the last six rows/columns, in the order. It is very convenient when you are defining a different style for the last rows/columns of your table but you do not know or do not want to count the number of rows/columns. Please be aware that the values `U`, `V`, and `W` were added in a very recent version of the package. Remember to update your TEX distribution to enjoy them, quack!

Box 2 – Column and row options

```
\begin{tblr}{colspec={llccc},
  rows={m, rowsep=2pt},
  columns={colsep=3pt},
  row{1,Z}={font=\bfseries, abovesep=3pt,
    belowsep=1pt},
  column{Z}={rightsep=0pt, fg=red},
  column{1}={leftsep=0pt},
  column{3-Z}={yellow!10}}
\hline
{First\\ name}&{Last\\ name}& A & B &
    Average\\\hline
Paulinho & van Duck & 10 & 20 & 15\\
Paulette & de la Quack & 30& 40 & 35
    \\\hline
Total && 40 & 60 & 50\\\hline
\end{tblr}
```

First name	Last name	A	B	Average
Paulinho	van Duck	10	20	15
Paulette	de la Quack	30	40	35
Total		**40**	**60**	**50**

Box 2 shows an example. Please note that when other parameters are present, the column alignment must be specified using `colspec`. The first and the last row font is set to bold (with `font=`⟨*font commands*⟩). Columns from the third to the last have a background color, which can simply be set with ⟨*color*⟩, without specifying the key `bg`. For the foreground color, on the contrary, the key `fg=`⟨*color*⟩ is mandatory. Please remember to also load `xcolor` package, and use your imagination if you are reading the black-and-white version of this article.

You can also note the multiline cells simply created with `{...\\...}`. Goodbye, `\makecell`!

For nearly all the parameters you can set in the mandatory argument there are corresponding commands you can use inside the table contents. For example, `\SetRow` is like `row`. For the sake of brevity, with few exceptions, I will not show these commands here, please refer to the manual [2] for them.

2.2 No more pain with vertical alignment

The package `tabularray` allows very easy alignment setting.

Box 3 – Vertical alignment with reference to baseline

```
\begin{tblr}{colspec={Q[l,t]Q[c,m]Q[r,b]},
    hlines}
{Baseline is\\ the top line\\ (left
    aligned)} &
{Baseline is\\ at the middle\\ (centered)} &
{Baseline is\\ the bottom line\\ (right
    aligned)}\\
\end{tblr}
```

Baseline is the top line (left aligned)	Baseline is at the middle (centered)	Baseline is the bottom line (right aligned)

Box 4 – Vertical alignment for non-typographers

```
\begin{tblr}{colspec={Q[l,h]Q[c,m]Q[r,f]},
    hlines}
{At the head\\ (left aligned)} &
{At the\\ middle\\ (centered)} &
{At the foot\\ (right aligned)}\\
\end{tblr}
```

At the head (left aligned)	At the middle (centered)	At the foot (right aligned)

Using its "Quack" column type `Q[`⟨*styles*⟩`]`, you can simultaneously indicate the vertical and horizontal alignments. No more "complex" options like `>{\centering\arraybackslash}` needed!

The vertical alignment can be set with respect to the baseline as usual, using `t` (the baseline is the top line), `m` (the baseline is at the middle), or `b` (the baseline is the bottom line).

An example is in Box 3 (the `hlines` option will be explained in Section 2.4).

Since the "baseline" concept is not easily caught by newbies, `tabularray` also allows the `h` (head) and `f` (foot) alignment, see Box 4.

Even more awesome is the possibility to combine the alignment as you like, as in Box 5. Please note that if you use the pipes `|` in `rowspec` they are horizontal lines.

2.3 Customizable cell dimensions

Within the styles of your column, you can set the column width; the option is `wd=`⟨*dimension*⟩, but `wd=` can often be omitted.

Box 5 – Combined horizontal/vertical alignment

```
\begin{tblr}{colspec={Q[l]Q[c]Q[r]},
  rowspec={|Q[t]|Q[m]|Q[b]|}}
{Top\\ left} & Top centered & Top right \\
Middle left & {Middle\\ centered} & Middle
    right \\
Bottom left & Bottom centered & {Bottom\\
    right} \\
\end{tblr}
```

Top left	Top centered	Top right
Middle left	Middle centered	Middle right
Bottom left	Bottom centered	Bottom right

The same is true for the cell height, with the analogous key `ht=`⟨*dimension*⟩.

These two parameters together allow you to create cells of the exact dimension you need, both horizontal and *vertical*. No more hacking with struts!

The sudoku scheme in Box 6 is an example of perfectly square cells. The `stretch=0` option is used to have the numbers perfectly centered.

The solution of the puzzle is left to the reader. I also have to mention a `sudoku` package [1] already exists, quack!

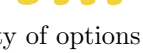
2.4 Lines as you desire

Box 6 is also an example of how you could easily customize table rules (a.k.a. lines).

With `hlines`/`vlines` you can draw with a single option all the horizontal/vertical rules. There is no more need to put `\hline` at the end of every line or a pipe `|` between the column types, although they are accepted as well.

The advantage of `tabularray` is that you can customize any given rule, or even part of one. With:
`hline{`⟨*hnumber*⟩`}={`⟨*vnumber*⟩`}{`⟨*styles*⟩`}`
you can specify for which horizontal rules ⟨*hnumber*⟩, and from/to which vertical rules ⟨*vnumber*⟩, your options apply. The ⟨*hnumber*⟩/⟨*vnumber*⟩ could be also a range ⟨*n–m*⟩ or a list ⟨*n,m,p,q*⟩. Please pay attention that, in this case, they are the *rule* numbers, not the row/column numbers as in `row` or `column`.

The analogous option for vertical rules is:
`vline{`⟨*vnumber*⟩`}={`⟨*hnumber*⟩`}{`⟨*styles*⟩`}`



There are plenty of options for setting the rule appearance: width (`wd=`⟨*dimension*⟩), shape (`solid`,

Box 6 – A sudoku puzzle

```
\begin{tblr}{columns={.5cm, colsep=0pt},
  rows={.5cm, rowsep=0pt},
  cells={c,m},hlines,vlines,stretch=0,
  hline{1,4,7,Z}={wd=1.2pt},
  vline{1,4,7,Z}={wd=1.2pt}}
1& &6& & &5&4& & \\
 & &9&1& &8& &5& \\
7&5& &9& & &1& &3\\
 & & &5& & &3& &9\\
2& & & &4& & & &1\\
8& &4& & &9& & & \\
5& &7& & &3& &2&6\\
 &6& &8& &1& &5& \\
 & &3&2& &9& &7&\\
\end{tblr}
```

1		6			5	4		
		9	1		8		5	
7	5		9			1		3
			5			3		9
2				4				1
8		4			9			
5		7			3		2	6
	6		8		1	5		
		3	2			9		7

`dashed`, `dotted`), color (`fg=`⟨*color*⟩). In Box 6, I use this to make the first and then every third rule (both horizontal and vertical) a little thicker.

But, in my opinion, the most awesome feature is the possibility to use `U`, `V`, `W`, `X`, `Y`, and `Z` to denote the last six lines, respectively (as was seen above for rows and columns, but now for rules). This allows you to completely customize your table within the mandatory parameter of the `tblr` environment without worrying about adding new rows/columns. It is very useful if you would like to create your own tabularray environment, see Section 2.8.

2.5 Multirow (and multicolumn) never so easy

In ordinary tabular environments, in the presence of multirow cells, the vertical alignment is often a pain in the . . . quack! One of the big strengths of `tabularray` is how it manages this alignment.

In traditional environments, when some cells have more than one line of text, it is often necessary to manually adjust the multirow text position. `tabularray` does it automatically: see Box 7.

Box 7 – Multirow comparison

```
\begin{tabular}{|p{2.1cm}|p{2.1cm}|}
\hline
\multirow{2}{*}{Multirow cell}& With
    \texttt{tabular}\\\cline{2-2}
& A cell with two text lines \\\hline
\end{tabular}\par\vspace{1ex}
\begin{tblr}{columns={2.1cm}, hlines,
  vlines, cell{1}{1}={r=2}{l}}
Multirow cell & With \texttt{tblr}\\
& A cell with two text lines \\
\end{tblr}
```

Multirow cell	With `tabular`
	A cell with two text lines

Multirow cell	With `tblr`
	A cell with two text lines

Please also note the horizontal lines are correctly drawn without specifying the columns where they should appear. The `multirow` package and `\cline` command are no longer needed!

The option to set a multirow/multicolumn is
`cell{`⟨*i*⟩`}{`⟨*j*⟩`}={`⟨*span*⟩`}{`⟨*styles*⟩`}`
where ⟨*i*⟩ and ⟨*j*⟩ are the row and column numbers (top left position of the multirow/multicolumn cell); ⟨*span*⟩ indicates how many rows (`r=`⟨*number*⟩) and/or how many columns (`c=`⟨*number*⟩) the cell should span; and ⟨*styles*⟩ gives the options of the multicell.

If you prefer, you can use an analogous command at the cell position: `\SetCell[`⟨*span*⟩`]{`⟨*styles*⟩`}`

When you merge cells horizontally, please note that you still need to give all necessary `&` characters for the empty cells. This is different behavior compared to the ordinary `\multicolumn`. If you omit any, the content of some of your cells might be missing. It can be difficult to detect, as it does not cause any error, only wrong output.

Another outstanding feature is the possibility of indicating the span algorithm.

For horizontal spanning, you can choose
`hspan=`⟨*algorithm*⟩
where ⟨*algorithm*⟩ can be `default` (the last column is enlarged to reach the width of the multicolumn cell), `even` (all the columns are equally enlarged to reach the width of the multicolumn cell), or `minimal`

Box 8 – Horizontal spanning

```
\begin{tblr}{hlines, vlines,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with default span \\
\end{tblr}
\par\vspace{1ex}
\begin{tblr}{hlines, vlines, hspan=even,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with even span\\
\end{tblr}\par\vspace{1ex}
\begin{tblr}{hlines, vlines, hspan=minimal,
  cell{2}{1}={c=3}{l}}
First & Second & Third \\
Multicolumn cell with minimal span\\
\end{tblr}
```

First	Second	Third
Multicolumn cell with default span		

First	Second	Third
Multicolumn cell with even span		

First	Second	Third
Multicolumn cell with minimal span		

(the multicolumn cell is split on more lines to fit the width of the columns). An example is in Box 8.

For vertical spanning, there is

`vspan=`⟨*algorithm*⟩

where ⟨*algorithm*⟩ could be `default` (the last row is stretched to reach the height of the multirow cell), or `even` (all the columns are equally enlarged to reach the length of the multicolumn cell). See Box 9.

2.6 Additional libraries

Even though `tabularray` has such a rich variety of features, you may well want to use it together with other traditional packages, such as `amsmath`, `booktabs`, or `siunitx`.

Since `tabularray` modifies some commands in those packages, to avoid potential conflict, you need to load them with `\UseTblrLibrary` command.

Box 10 shows an example with the `tabularray`'s libraries `booktabs` and `counter`. Unlike an ordinary table with `booktabs` rules, the background color touches them and the vertical rules are not discontinuous when they cross the horizontal ones.

The library `counter` allows you to modify some counters inside `tabularray` tables. In the example,

Box 9 – Vertical spanning

```
\begin{tblr}{hlines, vlines,
  cell{1}{1}={r=2}{c}}
{Multirow\\ cell\\ with\\ default span} &
    First\\
& Second \\
\end{tblr}\par\vspace{1ex}
\begin{tblr}{hlines, vlines, vspan=even,
  cell{1}{1}={r=2}{c}}
{Multirow\\ cell\\ with\\ even span} &
    First\\
& Second \\
\end{tblr}
```

Multirow cell with default span	First
	Second

Multirow cell with even span	First
	Second

I used the counter `\mycount` that is increased by 1 at every row of the table, headers excluded.

The package `tabularray` also provides some counters that can be used with no need to load the library. They are: `rowcount` (total number of rows), `rownum` (number of the current row), `colcount` (total number of columns), and `colnum` (number of the current column). The first two are used in the third column of the table in Box 10.

With the option `preto=`⟨*text*⟩ I avoided writing the commands for showing the counters at every row. I wrote them only in the mandatory argument, specifying with `cell` where they should be applied. Is it not great? Quack!

`preto` prepends text to the cell; to append text to the cell you can use `appto`. And if the content of your cell should be the argument of a command, you can use `cmd`.

2.7 `X` column type

Like `tabularx`, `tabularray` provides for

`X[`⟨*coeff*⟩`,`⟨*alignm*⟩`,`⟨*styles*⟩`]`

columns, but with extra gear. With the optional parameters, you can configure their alignment ⟨*alignm*⟩, their width ⟨*coeff*⟩, or other settings ⟨*styles*⟩.

The width is in a form of a multiplicative coefficient, as in the (outdated) package `tabu`. For example, `X[2,r]` is a right-aligned column with a

Box 10 – booktabs and counter libraries

```
...
\usepackage{tabularray}
\UseTblrLibrary{booktabs}
\UseTblrLibrary{counter}
\newcounter{mycount}
\newcommand{\myc}{%
  \stepcounter{mycount}\arabic{mycount}}
...
\begin{document}
\begin{booktabs}{vlines,
  colspec = {lcc}, hspan=even,
  cell{1}{1} = {r=2}{},
  cell{1}{2} = {c=2}{c},
  row{odd[3]} = {bg=cyan!10},
  cell{3-Z}{2}={preto={\arabic{rownum} of
    \arabic{rowcount}}},
  cell{3-Z}{3}={preto={\myc}}}
\toprule
Name & Counters \\
\cmidrule{2-3}
& {\texttt{rownum} of\\ \texttt{rowcount}}
    & \texttt{mycount} \\
\midrule
Paulinho van Duck & & \\
Paulette de la Quack & & \\
Paolino Quaqua & & \\
Pauline von Ente & & \\
Paulina Pato & & \\
\bottomrule
\end{booktabs}
\end{document}
```

Name	Counters	
	`rownum` of `rowcount`	`mycount`
Paulinho van Duck	3 of 7	1
Paulette de la Quack	4 of 7	2
Paolino Quaqua	5 of 7	3
Pauline von Ente	6 of 7	4
Paulina Pato	7 of 7	5

doubled width, compared to the other `X[1]` columns of the table (`[1]` is the default and can be omitted).

`X` columns with negative coefficients are also possible. In this case, the columns have their "natural width", but are limited to the width of a corresponding `X` column with a positive coefficient. For instance, `X[-1]` is like an `l` column but if its width is greater than what an `X[1]` column would have had, then it is adapted to the `X[1]` column width. In Box 11, column C has the same width as E or F, whereas column D has its natural width.

Box 11 – X column type

```
\begin{tblr}{width={.9\linewidth},
  colspec={X[2, font={\itshape}]
    X[2,c]X[-1]X[-1]XX[r]},
  vlines, hlines}
A & B & C & D & E & F\\
Q & u & a a a  & c & k & !!!\\
\end{tblr}
```

A	B	C	D	E	F
Q	u	a a a	c	k	!!!

The total width of the table is `\linewidth` by default. If you need a different width, use the option `width=`⟨*dimension*⟩.

2.8 New row/column types, new environments, and other tricks

As for ordinary environments, you can create your own column types with:
`\NewColumnType{`⟨*type*⟩`}[`⟨*n*⟩`][`⟨*dflt*⟩`]{`⟨*styles*⟩`}`
where ⟨*type*⟩ is one letter indicating the name of the new column type; ⟨*n*⟩ is the number of parameters, if any; ⟨*dflt*⟩ is the default parameter, if any; ⟨*styles*⟩ are the column options.

Since, with `tabularray`, you have row options, you can also create your own row types, with the analogous command `\NewRowType`.

In Box 12, column type `T` is defined as left aligned, text mode (`mode=text` allows to use a text column in a math environment); and `D` as centered display math (`dmath`) mode. The option `mode` also provides for `imath` and `math` for inline math mode.

As mentioned above, the possibility to have all the settings in the mandatory argument is particularly useful if you like to create your own table template, because styles are totally separated from the contents of your tables.

With `\NewTblrEnviron{`⟨*envname*⟩`}` you can define your own environment. Then, with
`\SetTblrInner[`⟨*envname*⟩`]{`⟨*styles*⟩`}`
you can set all the styles you like for the environment ⟨*envname*⟩. Here the environment name is optional; if you leave it out, the styles for all the `tblr` of your document will be set.

For example, `\SetTblrInner{rowsep=0pt}` sets the spacing as in the ordinary `tabular`, for all your `tblr` environments.

Box 12 – New types and environment

```
\NewColumnType{T}{Q[l, mode=text]}
\NewColumnType{D}{Q[c, mode=dmath]}
\NewTblrEnviron{mytab}
\SetTblrInner[mytab]{colspec={TD},
  row{1,Z}={font=\bfseries, mode=text},
  hline{1,2,Y,Z}={leftpos=-1, rightpos=-1,
    endpos},
        cells={m}}
\begin{mytab}{}
  Shape&Area\\
  Circle & \pi r^2 \\
  {Total rows of this table} &
    \arabic{rowcount}\\
\end{mytab}\par\vspace{1ex}
\begin{mytab}{rowsep=3pt}
  Shape&Area\\
  Square & a^2 \\
  Rectangle & w\cdot h\\
  {Total rows of this table}  &
    \arabic{rowcount}\\
\end{mytab}
```

Shape	**Area**
Circle	πr^2
Total rows of this table	**3**

Shape	**Area**
Square	a^2
Rectangle	$w \cdot h$
Total rows of this table	**4**

There is also
`\SetTblrOuter[`⟨*envname*⟩`]{`⟨*styles*⟩`}`
to set the specifications of the optional argument of your environment, such as the baseline. Please refer to the package manual [2] for this.

Box 12 shows an example with the new environment `mytab`. Please note the use of the `Y` and `Z` in the `hline` and `row` options. This way you build a template that is always valid, regardless of the number of rows in the table.

The options `leftpos=-1` and `rightpos=-1` of `hline` mean that the lines are trimmed by `colsep` on the left and on the right; with `endpos` this adjustment is applied only to the leftmost/rightmost column.

The `tabularray` support for having all the definitions separated from the content turns out to be very helpful when you have to format tables for which you do not have easy access to the source code, e.g. because they are automatically generated by R or Markdown or another tool.

Box 13 – A useful trick

```
\documentclass{article}
\usepackage{tabularray}
\renewenvironment{tabular}[2][c]{
  \begin{tblr}[baseline=#1]{
    row{1-2}={font=\bfseries},
    colspec={#2}}
}{\end{tblr}}
\begin{document}
\begin{table}[htbp]
\centering
\input{tab.tex}
\end{table}
\end{document}
```

Imagine having the code of a table you cannot change, such as `tab.tex`, but you would like to have the first two rows in bold. With `tabularray` you could redefine the `tabular` environment adding any style you like. Box 13 shows how to do it.

3 Conclusions

I hope you liked the features supplied by this modern package, and if you have problems with your tables, remember:

Try `tabularray`
and all will be OK!

References

[1] P. Abraham. The sudoku package. Version 1.0.1. `ctan.org/pkg/sudoku`

[2] J. Lyu. Tabularray — Typeset Tabulars and Arrays with LaTeX3. Version 2023A. `ctan.org/pkg/tabularray`

[3] F. Sihler. The Ti*k*Zpingus package. Version 1.0. `ctan.org/pkg/tikzpingus`

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond
Milano, Italy
`paulinho dot vanduck (at) gmail dot com`