tabularcalc

v0.1

User's manual

Christian Tellechea
unbonpetit@gmail.com

March 19th 2009

Abstract

Given a list of numbers and one (or more) formulas, this package allows with an easy syntax to build a table of values, i.e a tables in which the first row contains the list of numbers, and one (or more) others rows contain the *calculated* values of formulas for each number of the list:

x	-4	-2	0	2.25	7
$f(x)=2x-3$	-11	-7	-3	1.5	11
x^2	16	4	0	5.062 5	49
$h(x)=\sqrt{x^2+1}$	4.123 1	2.236 05	1	2.462 2	7.071 06

The table can be built either horizontally or vertically, and it is fully customizable (height of rows, columns and lines types). Moreover, the content of any cell can be easily hidden.

Other local effects are possible since a command allows to execute any code in any particular cell.

Contents

Attention: this manual is the laboured[1] translation of the french manual.

1 Introduction

This package needs LaTeX2ε, and if it has not been done before, loads the following packages: pgfmath, xstring and numprint. It makes easily possible to build tables of calculated results coming from formulas for a given list of values. Tables are displayed using the standard tabular[2] environment. The package is called with the usual command \usepackage{tabularcalc}

This package is not intended to compete with the excellent pgfplotstable package of Christian Feuersänger which has much more extended features, but in compensation, has a difficult to learn syntax. tabularcalc is meant to be more modest and gives priority to customization and easy syntax.

For calculation, the computation of an arithmetic expression such as 2*x*x-5*x+7 when x = 2.7 is with TeX, a very complex thing that tabularcalc does not make. It leaves this task to a computation engine: "pgfmath" by default. This is why pgfmath is loaded. It is possible to choose an other computation engine, see page 9.

To display decimal numbers, in my view, nothing is better than the numprint package. This is why it is loaded. The engine used to display decimal numbers can be changed or customized, see page 7.

To define vocabulary for later use, in the simple tables below, red numbers are the "values", blue numbers are the "results" and brown texts are the "labels". The cell on the up-left corner is the "cell(0,0)":

[1] Indeed, I **do not speak english**, and I did my best to achieve this translation. Please, be indulgent, and try to take my place and imagine what it would be for you if you had to translate a manual into french, with some old poor school knowledge!

[2] For the moment, the tabular environment is hard coded, but it will probably be possible in the next version to let the user choose another table environment: tabularx, tabulary, supertabular, etc.

Horizontal table

cell (0,0)	−5	−1	0	3	10
x	−5	−1	0	3	10
$2x$	−10	−2	0	6	20
$3x$	−15	−3	0	9	30

Vertical table

cell (0,0)	x	$2x$	$3x$
−5	−5	−10	−15
−1	−1	−2	−3
0	0	0	0
3	3	6	9
10	10	20	30

2 Basic features

2.1 Horizontal tables

The macro \htablecalc builds horizontal table whose first row contains the "values" and the other rows the "results". The syntax is:

\htablecalc[⟨*number*⟩]{⟨*cell (0,0)*⟩}{⟨*value list*⟩}
{⟨*label 1*⟩}{⟨*formula 1*⟩}
{⟨*label 2*⟩}{⟨*formula 2*⟩}
...
{⟨*label n*⟩}{⟨*formula n*⟩}

where :

- ⟨*number*⟩ is the number of formulas (1 by default);
- ⟨*cell (0,0)*⟩ is the content of the cell (0,0);
- ⟨*value list*⟩ is the list of values, separated with a comma. Two consecutive commas make an empty column;
- ⟨*label i*⟩ is the i$^{\text{th}}$ label;
- ⟨*formula i*⟩ is the i^{th} formula, used to calculate the reults of the i^{th} row. In formulas, `x` is the variable.

The variable "`x`" is the expansion of \numberletter, and at any moment, it is possible to redefine it to another letter, "`y`" for example with a \def\numberletter{y}.

In the list of values, a comma separate values by default. This comma is the expansion of \listsep, and can be changed to "|" for example with \def\listsep{|}

For a first example, here is a try to obtain the table of the first page:

```
\htablecalc[3]{$x$}{-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

x	−4	−2	0	2.25	7
$f(x) = 2x - 3$	−11.0	−7.0	−3.0	1.5	11.0
x^2	16.0	4.0	0.0	5.062 5	49.0
$h(x) = \sqrt{x^2 + 1}$	4.123 1	2.236 05	1.0	2.462 2	7.071 06

This table is not strictly the same than the table of the first page: integer results have a "0" as decimal part, columns containing results do not have the same width and the line at the bottom of the first row is different.

2.2 Vertical tables

The macro \vtablecalc builds vertical table whose first column contains the "values" and the other rows the "results". The syntax is:

\vtablecalc[⟨*number*⟩]{⟨*cell (0,0)*⟩}{⟨*value list*⟩}
{⟨*label 1*⟩}{⟨*formula 1*⟩}
{⟨*label 2*⟩}{⟨*formula 2*⟩}
...
{⟨*label n*⟩}{⟨*formula n*⟩}

where :

- ⟨*number*⟩ is the number of formulas (1 by default);
- ⟨*value list*⟩ is the list of values, separated with a comma;
- ⟨*label i*⟩ is the i^{th} label;
- ⟨*formula i*⟩ is the i^{th} formula, used to calculate the reults of the i^{th} column.

Here is the previous table, but vertically built:

```
\vtablecalc[3]{$x$}{-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

x	$f(x) = 2x - 3$	x^2	$h(x) = \sqrt{x^2 + 1}$
-4	-11.0	16.0	4.123 1
-2	-7.0	4.0	2.236 05
0	-3.0	0.0	1.0
2.25	1.5	5.062 5	2.462 2
7	11.0	49.0	7.071 06

2.3 How to hide numbers

The content of any cell can be hidden, as well as in a horizontal or vertical table.

2.3.1 Hide a value

In the list of values, a "@" before a value hides it. In the following example, the second and fifth values are hidden:

```
\htablecalc[3]{$x$}{-4,@-2,0,2.25,@7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

x	-4		0	2.25	
$f(x) = 2x - 3$	-11.0	-7.0	-3.0	1.5	11.0
x^2	16.0	4.0	0.0	5.062 5	49.0
$h(x) = \sqrt{x^2 + 1}$	4.123 1	2.236 05	1.0	2.462 2	7.071 06

Behind the scene, the "@" token is the expansion of \noshowmark. To change this token to "=", this simple code does the job: \def\noshowmark{=}

2.3.2 Hide a result

If a value is followed by $[a_1][a_2]\dots[a_n]$ where the numbers a_i are increasing, the results number $a_1, a_2, \dots, a_n$ will be hidden. If a number $a_j = 0$, all the others a_k where $k > j$ will be ignored and the results following the previous hidden result will be hidden.

In the example, with the list of values "`-4[2],-2,0[1][3],2.25[0],7[2][0]`", we are going to:

- hide the second result of the first value with "`-4[2]`"
- let all the results visible for the second value with "`-2`"
- hide the results number 1 and 3 of the third value with "`0[1][3]`"
- hide all the results of the fourth value with "`2.25[0]`"
- for the fifth value, hide all the results from the second with "`7[2][0]`"

```
\htablecalc[3]{$x$}{-4[2],-2,0[1][3],2.25[0],7[2][0]}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

x	-4	-2	0	2.25	7
$f(x) = 2x - 3$	-11.0	-7.0			11.0
x^2		4.0	0.0		
$h(x) = \sqrt{x^2 + 1}$	4.123 1	2.236 05			

This feature can be mixed with "`@`" to hide a value and results.

2.4 Height of rows

At the begining of a row, when it is displayed, the macro `\startline` runs.
By default, this command is defined by: `\def\startline{\rule[-1.2ex]{0pt}{4ex}}`. Its expansion is a "strut" which adjusts the height of the row. Here is this strut, made visible before the lettre "a":
a

Any other action, or another strut can be defined:

```
\def\startline{%
    {\bfseries\number\tclin)\ }%
}
\htablecalc[3]{$x$}{-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

0) x	-4	-2	0	2.25	7
1) $f(x) = 2x - 3$	-11.0	-7.0	-3.0	1.5	11.0
2) x^2	16.0	4.0	0.0	5.062 5	49.0
3) $h(x) = \sqrt{x^2 + 1}$	4.123 1	2.236 05	1.0	2.462 2	7.071 06

Here, no strut is defined (the lines recover their natural height), and at line 2 of the code, the number of the row (contained in the counter `\tclin`) is displayed with bold chars.

2.5 Horizontal lines

tabularcalc allows to define 3 types of horizontal lines. The macro \sethrule has 3 arguments:

- the first that we call "line 0" is displayed on the top and bottom of the table;
- the second, "line 1", is displayed at the bottom of the first row;
- the third, "other lines", is displayed at the bottom of the other rows, excepted the last one which is the bottom of the table.

Here is the syntax:
\sethrule{⟨*line 0*⟩}{⟨*line 1*⟩}{⟨*other lines*⟩}

By default, the three arguments contain \hline.

This is an example in which the "line 1" is a double line, and the "other lines" are not drawn:

```
\sethrule{\hline}{\hline\hline}{}
\htablecalc[3]{$x$}{-2,-1,0,1,2,3}
              {$2x$}{2*x}
              {$3x$}{3*x}
              {$4x$}{4*x}
```

x	-2	-1	0	1	2	3
$2x$	-4.0	-2.0	0.0	2.0	4.0	6.0
$3x$	-6.0	-3.0	0.0	3.0	6.0	9.0
$4x$	-8.0	-4.0	0.0	4.0	8.0	12.0

The command \resethrule resets the defined lines and restores the default lines.

2.6 Customizing columns

2.6.1 Vertical lines

2 types of column can be defined: the type of the left one and the type of others columns. The command \setcoltype has an optionnal argument and 2 mandatory arguments:

- the optional argument, empty by default, defines the vertical lines at the right of the table;
- the "type 1" of the first column, set to "|c|" by default;
- the "type 2" of the other columns, set to "c|" by default.

The syntax of the command is:
\setcoltype[⟨*right lines*⟩]{⟨*type 1*⟩}{⟨*type 2*⟩}

In this example, a double line is displayed at the right of the table ([||]), and on the edges of the first column (||c||). The other columns do not have vertical lines (c):

```
\setcoltype[||]{||c||}{c}
\htablecalc[3]{$x$}{-2,-1,0,1,2,3}
              {$2x$}{2*x}
              {$3x$}{3*x}
              {$4x$}{4*x}
```

x	-2	-1	0	1	2	3
$2x$	-4.0	-2.0	0.0	2.0	4.0	6.0
$3x$	-6.0	-3.0	0.0	3.0	6.0	9.0
$4x$	-8.0	-4.0	0.0	4.0	8.0	12.0

\resetcoltype restores the default vertical lines.

2.6.2 Width of columns

Instead of the usual column type "c" used until now, other types of column can be specified: for example, the "m" type of the array package allows to set the width of columns this way: m{1.5cm}.

In this example, the first column is right aligned, and the other columns are centered and 1.5 cm width:

```
\usepackage{array}
\setcoltype{|r|}{>{\centering\arraybackslash}m{1.5cm}|}
\htablecalc[3]{$x$}{-4,-2,0,2.25,7}
                  {$f(x)=2x-3$}{2*x-3}
                  {$x^2$}{x*x}
                  {$h(x)=\sqrt{x^2+1}$}{sqrt(x*x+1)}
```

x	-4	-2	0	2.25	7
$f(x) = 2x - 3$	-11.0	-7.0	-3.0	1.5	11.0
x^2	16.0	4.0	0.0	5.062 5	49.0
$h(x) = \sqrt{x^2 + 1}$	4.123 1	2.236 05	1.0	2.462 2	7.071 06

3 Advanced customization

3.1 How to execute a code in a cell

The command \defcellcode allows to execute any code in a unique cell, or in every cells of a row or in every cells of a column. Cells have the following coordinates:

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)

Here is the syntax:
\defcellcode{⟨*number 1*⟩}{⟨*number 2*⟩}{⟨*code*⟩}
where :

- ⟨*number 1*⟩ is the first coordinate (row number);
- ⟨*nombre 2*⟩ is the second coordinate (column number);
- ⟨*code*⟩ is the code executed *when the specified cell is displayed*;
- if ⟨*number 1*⟩ is empty, all the rows are concerned;
- if ⟨*nombre 2*⟩ is empty, all the columns are concerned;

Behind the scene, the first coordinate – the row number – is the counter \tclin, and the number of the column is the counter \tccol.

Notice that the code is expanded when the cell is displayed, and at that moment, the counter \tccol does not contain anymore the column number of the cell: you should **not** use \tccol in the code definied with the macro \defcellcode. On the other hand, the counter \tclin does contain the reliable number of the current line.
If codes are defined with \defcellcode and several of them are runned in the same cell, they will be executed in the same order of their definition.

In this example, with the package xcolor, the cell (2 , 3) is colored in blue, the row 1 in red and the column 4 in brown:

```
\usepackage{color}
\defcellcode{2}{3}{\color{blue}}
\defcellcode{1}{}{\color{red}}
\defcellcode{}{4}{\color{brown}}
\htablecalc[3]{$x$}{-2,-1,0,1,2,3}
              {$2x$}{2*x}
              {$3x$}{3*x}
              {$4x$}{4*x}
```

x	-2	-1	0	1	2	3
$2x$	-4.0	-2.0	0.0	2.0	4.0	6.0
$3x$	-6.0	-3.0	0.0	3.0	6.0	9.0
$4x$	-8.0	-4.0	0.0	4.0	8.0	12.0

Notice that the cell (1 , 4) whose content is 2.0 has been colored in red (line 3 of the code) *and then* in brown (line 4 of the code).

Another similar command is provided to execute code in a cell: `\edefcellcode`. With this command, the code is expanded a first time with an `\edef`[3] when cell is built: at this time, the counter `\tccol` does contain the number of the column. Then, the expansion obtained is runned a second time when cell is displayed.

In this example, text is blue if the column number is greater than 2:

```
\usepackage{color}
\edefcellcode{}{}{%
    \ifnum\tccol>2 \noexpand\color{blue}\fi}
\htablecalc[3]{$x$}{-2,-1,0,1,2,3}
              {$2x$}{2*x}
              {$3x$}{3*x}
              {$4x$}{4*x}
```

x	-2	-1	0	1	2	3
$2x$	-4.0	-2.0	0.0	2.0	4.0	6.0
$3x$	-6.0	-3.0	0.0	3.0	6.0	9.0
$4x$	-8.0	-4.0	0.0	4.0	8.0	12.0

3.2 Customizing the number display

3.2.1 Macros `\printvalue` and `\printresult`

To display a value, the macro `\printvalue` is called. It requires one argument: the number to display which comes from `pgfcalc`. This argument has a raw format: `12345.6789` for "12,345.678 9".
By default, `\printvalue` is defined with this code:

```
\def\printvalue#1{\numprint{#1}}
```

Notice that the macro `\numprint` is called to print the number.

To display a result, the macro `\printresult` is called. It requires **two** arguments: the first is the number to display in raw format coming from `pgfcalc` and the second is the value used to compute the result.
By default, `\printresult` is defined with this code:

```
\def\printresult#1#2{\numprint{#1}}
```

Notice that the argument `#2` (the value) is ignored by `\printresult`. But it is easy to imagine an example in which it would not be. In this example, a red "`X`" is printed if the lenght of the square

[3] If a command must not be expanded at this time, a `\noexpand` must be put before it.

(which is argument #2) is negative. If not, the result with the unit is printed. For the pleasure of customization, any result less than 10 is printed in blue:

```
\usepackage{color}
\def\printresult#1#2{%
    \ifdim#1pt<10pt\color{blue}\fi
    \ifdim#2pt<0pt
        \color{red}\texttt{X}%
    \else
        \numprint[cm^2]{#1}%
    \fi}
\htablecalc{length}{0.7,-10,3,-2,5,12}
           {Area of square}{x*x}
```

length	0.7	−10	3	−2	5	12
Area of square	0.489 99 cm^2	X	9.0 cm^2	X	25.0 cm^2	144.0 cm^2

A remark: when the length is 0.7 cm, the result is slightly wrong. It should be 0.49 cm^2 instead of 0.489 99 cm^2! The pgfmath package is not suitable for scientific computation as it is inteded to compute coordinates for drawing purposes. This is why its precision is $\frac{1}{100,000}$ and sometimes leads to results including rounding errors such as this one.

3.2.2 How to control the rounding of numbers

With integer results, the pgfmath package, though excellent, has a annoying drawback: when the result of the computation is an integer, the returned number has a decimal part "0" (see table above). To avoid this, the result can be tested with \IfInteger of xstring package, and if it is an integer, give to \numprint the value of the \integerpart counter (see xtring donumentation):

```
\def\printresult#1#2{%
    \IfInteger{#1}%
        {\numprint{\number\integerpart}}%
        {\numprint{#1}}%
}
\htablecalc{$x$}{-3,1.56,2.5,3.608}{$2x$}{2*x}
```

x	−3	1.56	2.5	3.608
$2x$	−6	3.12	5	7.216

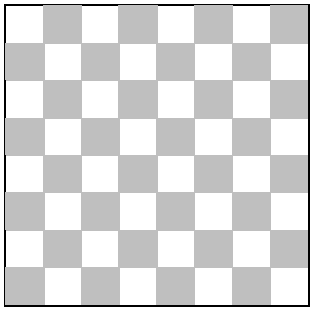
It is also possible to force numprint to round its argument with the command \nprounddigits whose argument is the number of figures of the decimal part. Unfortunately, if needed, unnecessary 0 are added to fill the decimal part to match the number of figures.

3.2.3 For the fun

Other uses of this package can be designed, such as the drawing of a chess board which squares are 0.5 cm long:

- on line 2, the separators of the table are initialzes at 0pt to obtain the length of 0.5 cm;
- the display of values and results is cancelled at line 3;
- horizontal lines of the top and bottom of the table are drawn (line 4), and vertical lines of the left and right (line 5);
- a strut 0.5 cm height is defined to be displayed at the begining of every row (line 6);
- finally, if the sum of the row number and the column number is odd, the square is filled of gray (line 8 and 9).

```
\usepackage{colortbl,xcolor}
\arraycolsep=0pt\tabcolsep=0pt
\def\printvalue#1{}\def\printresult#1#2{}
\sethrule{\hline}{}{}
\setcoltype[|]{|m{0.5cm}}{m{0.5cm}}
\def\startline{\rule[-0.2cm]{0pt}{0.3cm}}
\edefcellcode{}{}{%
    \ifodd\numexpr\tccol+\tclin\relax
        \noexpand\cellcolor{lightgray}\fi
}
\htablecalc[7]{}{ , , , , , , }
    {}{}{}{}{}{}{}{}{}{}{}{}{}{}
```



4 How to change the computation engine

`pgfmath` is the computation engine used by default, but it can be changed though there is not many other choices: as far as I know, `fp` and `xlop` are able to compute math expression under LaTeX. Both have drawbacks, this is why they are not used by default:

- "`fp`" uses the Reverse Polish Notation (RPN). It is possible to use the infix notation but the opposite sign " − " before a number is not accepted: very annoying for negative values!
- "`xlop`" is not really a computation package as it also prints its results. And unfortunately, only arithmetic calculation is possible.

The macro `\tc@evalexpr` is in charge of computation. It has no argument and it works like this: it expands `\tc@currentresult` whose expansion is the math expression to compute (for example "`4*2.5*2.5-3*2.5-7`"). It calls the computation engine to achieve the calculation (in this example, it would be `10.5`), and finally it assigns this result to `\tc@currentresult`.
By default, the macro `\tc@evalexpr` is defined with this code:

```
\def\tc@evalexpr{%
    \expandafter\pgfmathparse
    \expandafter{\tc@currentresult}%
    \let\tc@currentresult\pgfmathresult}
```

Let us suppose we want to to use the `fp` package for the computation. Let us redefine `\tc@evalexpr`:

```
\makeatletter
\def\tc@evalexpr{%
    \expandafter\FPeval\expandafter
    \tc@currentresult\expandafter{\tc@currentresult}%
    \FPclip\tc@currentresult\tc@currentresult
}
\makeatother
```

The command `\FPclip` removes unnecessary `0` from the result (see `fp` documentation).
We have done it: the computation engine of tabularcalc is now `fp`:

```
\usepackage{fp}
\htablecalc{$x$}{0.5,1,2.5,10}
    {$2x^2-\frac{1}{2}$}{2*x*x-0.5}
```

x	0.5	1	2.5	10
$2x^2-\frac{1}{2}$	0	1.5	12	199.5

An issue remains: as `fp` does not understand the opposite " − " sign before numbers, and a value like `"-3"` would have provoked an error! To circumvent this, the RPN can be used, but `\tc@evalexpr` must be modified to tell it that the computation has to be done using the RPN (this is why `\FPupn` is used). Also, `\printvalue` must be modified to make it compute with `\FPupn` the values also written in RPN (`3 neg` in the list of values instead of `-3`):

```
\usepackage{fp}
\makeatletter
\def\tc@evalexpr{%
    \expandafter\FPupn\expandafter
    \tc@currentresult\expandafter{\tc@currentresult}%
    \FPclip\tc@currentresult\tc@currentresult
}
\makeatother
\def\printvalue#1{%
    \FPupn\tempval{#1}%
    \FPclip\tempval\tempval
    \numprint\tempval
}
\htablecalc{$x$}{3 neg,0.5,1,2.5,10}
    {$2x^2-\frac{1}{2}$}{x x mul 2 mul 0.5 sub}
```

x	-3	0.5	1	2.5	10
$2x^2-\frac{1}{2}$	17.5	0	1.5	12	199.5

⋆
⋆ ⋆

That's all, I hope you will find this package useful!
Please, send me an email if you find a bug or if you have any idea of improvement...

Christian Tellechea