tabularcalc

v0.2

User's manual

Christian Tellechea
unbonpetit@gmail.com

April 21st 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 the other rows contain the *calculated* values of the 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
$h(x)=\sqrt{x^2+1}$	$4.123\,106$	$2.236\,068$	1	$2.462\,214$	$7.071\,068$

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.

Many thanks to Derek O'CONNOR for the interest he brought to this package and the tests he made on beta versions. His pertinent suggestions of new features have been very useful. Without his advice, tabularcalc would not be what it is.

My thanks also to Le HUU DIEN KHUE who offered the translation of this manual into Vietnamese.

1 Introduction

1.1 Presentation

This package provides commands which make easy possible to build tables of calculated results coming from formulas for a given list of values. Tables are displayed using the standard `tabular` environment. tabularcalc needs LaTeX2ε and requires `fp`, `xstring` and `numprint` packages.

This package is not intended to compete with the excellent `pgfplotstable` package of Christian FEUERSÄNGER which has much more extended customization 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.

To display decimal numbers, in my view, nothing is better than the `numprint` package. The engine used to display decimal numbers can be changed or customized, see page 10.

---

[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!

1.2 The `fp` package

For calculation, the computation of an 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 math engine provided by the `fp` package. It provides all usual arithmetic, trigonometric and scientific operations. Moreover, infix and postfix notation are available: see the **`README`** file for the list of functions available for each notation.

I fixed 2 issues in the macro `\FPpow` of the `fp` package[2]. This macro is in charge of power calculation such as a^b.

- first of all a spurious space appears when a power is computed. This space is fixed by tabularcalc
- but there is another annoying issue: when `fp` computes a^b it uses this formula $a^b = \mathrm{e}^{b \ln a}$. There is an issue when b is an integer and a is negative. For example: $(-3)^2 = \mathrm{e}^{2 \ln(-3)}$. The logarithm of a negative number is undefined and `fp` is unable to compute this simple calculation. This bug is fixed and `fp` now computes this kind of calculation properly.

To enable tabularcalc fix these issues, the option "`fixFPpow`" can be declared when calling the package:

```
\usepackage[fixFPpow]{tabularcalc}
```

1.3 What is new?

Unfortunaltely, there are some incompatibilities with other packages because the name of macros of tabularcalc was already used. I decided with a heavy heart to rename almost all the public macros, risking a probable incompatibility with the previous version. I apologize for this inconvenience. I rename them with "`tc`" at their begining:

Old name	New name
`\noshowmark`	`\tcnoshowmark`
`\startline`	`\tcatbeginrow`
`\resetcellcode`	`\tcresetcellcode`
`\listsep`	`\tclistsep`
`\printvalue`	`\tcprintvalue`
`\printresult`	`\tcprintresult`
`\sethrule`	`\tcsethrule`
`\resethrule`	`\tcresethrule`
`\setcoltype`	`\tcsetcoltype`
`\resetcoltype`	`\tcresetcoltype`

Here is the other new features for the users of the previous version:

- calculation is made with `fp` since `pgfmath` has a poor precision;
- values can be computed;
- the code of a table can be exported in a file.

1.4 Vocabulary

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)":

---

[2] I did not warn the author of `fp` and I did not ask his permission beacuse he does not maintain his package for a long time now.

Horizontal table

cellule (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

cellule (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`[⟨*n*⟩]{⟨*cell (0,0)*⟩}{⟨*variable=list of values*⟩}
{⟨*label 1*⟩}{⟨*formula 1*⟩}
{⟨*label 2*⟩}{⟨*formula 2*⟩}
...
{⟨*label n*⟩}{⟨*formula n*⟩}

where :

- ⟨*n*⟩ is the number of formulas (1 by default);
- ⟨*cell (0,0)*⟩ is the content of the cell (0,0);
- ⟨*variable*⟩ is the dummy variable in ⟨*formula i*⟩ used to compute the results;
- ⟨*list of values*⟩ is the list of values, separated with a comma. Two consecutive commas make an empty column;
- ⟨*label i*⟩ is the i^{th} label;
- ⟨*formula i*⟩ is the i^{th} formula, used to calculate the reults of the i^{th} row.

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

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

```
\htablecalc[3]{$x$}{x=-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{round(root(2,x*x+1),6)}
```

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
$h(x)=\sqrt{x^2+1}$	$4.123\,106$	$2.236\,068$	1	$2.462\,214$	$7.071\,068$

This table is not strictly the same than the table of the first page: columns containing results do not have the same width and the line at the bottom of the first row is different. We will see later how to customize this.

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`[⟨*n*⟩]{⟨*cell (0,0)*⟩}{⟨*variable=list of values*⟩}
{⟨*label 1*⟩}{⟨*formula 1*⟩}
{⟨*label 2*⟩}{⟨*formula 2*⟩}
. . .
{⟨*label n*⟩}{⟨*formula n*⟩}

where :

- ⟨*n*⟩ is the number of formulas (1 by default);
- ⟨*cell (0,0)*⟩ is the content of the cell (0,0);
- ⟨*variable*⟩ is the dummy variable in ⟨*formula i*⟩ used to compute the results;
- ⟨*list of values*⟩ is the list of values, separated with a comma. Two consecutive commas make an empty column;
- ⟨*label i*⟩ is the i^{th} label;
- ⟨*formula i*⟩ is the i^{th} formula, used to calculate the reults of the i^{th} row.

Here is the previous table, but vertically built:

```
\vtablecalc[3]{$x$}{y=-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*y-3}
              {$x^2$}{y*y}
              {$h(x)=\sqrt{x^2+1}$}{round(root(2,y*y+1),6)}
```

x	$f(x) = 2x - 3$	x^2	$h(x) = \sqrt{x^2+1}$
-4	-11	16	4.123 106
-2	-7	4	2.236 068
0	-3	0	1
2.25	1.5	5.062 5	2.462 214
7	11	49	7.071 068

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]{$x$}{x=-4,@-2,0,2.25,@7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{round(root(2,x*x+1),6)}
```

x	-4		0	2.25	
$f(x) = 2x - 3$	-11	-7	-3	1.5	11
x^2	16	4	0	5.062 5	49
$h(x) = \sqrt{x^2+1}$	4.123 106	2.236 068	1	2.462 214	7.071 068

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

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]{$x$}{x=-4[2],-2,0[1][3],2.25[0],7[2][0]}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{round(root(2,x*x+1),6)}
```

x	-4	-2	0	2.25	7
$f(x) = 2x - 3$	-11	-7			11
x^2		4	0		
$h(x) = \sqrt{x^2+1}$	4.123 106	2.236 068			

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 `\tcatbeginrow` runs.
By default, this command is defined by: `\def\tcatbeginrow{\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:

```
1 \def\tcatbeginrow{%
2     {\bfseries\number\tclin)\ }%
3 }
4 \htablecalc[3]{$x$}{x=-4,-2,0,2.25,7}
5               {$f(x)=2x-3$}{2*x-3}
6               {$x^2$}{x*x}
7               {$h(x)=\sqrt{x^2+1}$}{round(root(2,x*x+1),6)}
```

0) x	-4	-2	0	2.25	7
1) $f(x) = 2x - 3$	-11	-7	-3	1.5	11
2) x^2	16	4	0	5.062 5	49
3) $h(x) = \sqrt{x^2+1}$	4.123 106	2.236 068	1	2.462 214	7.071 068

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 `\tcsethrule` 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:
`\tcsethrule`{⟨*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:

```
\tcsethrule{\hline}{\hline\hline}{}
\htablecalc[3]{$x$}{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	-2	0	2	4	6
$3x$	-6	-3	0	3	6	9
$4x$	-8	-4	0	4	8	12

The command `\tcresethrule` 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 `\tcsetcoltype` 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:
`\tcsetcoltype`[⟨*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`):

```
\tcsetcoltype[||]{||c||}{c}
\htablecalc[3]{$x$}{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	-2	0	2	4	6
$3x$	-6	-3	0	3	6	9
$4x$	-8	-4	0	4	8	12

`\tcresetcoltype` 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}
\tcsetcoltype{|r|}{>{\centering\arraybackslash}m{1.5cm}|}
\htablecalc[3]{$x$}{x=-4,-2,0,2.25,7}
              {$f(x)=2x-3$}{2*x-3}
              {$x^2$}{x*x}
              {$h(x)=\sqrt{x^2+1}$}{round(root(2,x*x+1),6)}
```

$x$	-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
$h(x) = \sqrt{x^2+1}$	4.123 106	2.236 068	1	2.462 214	7.071 068

3 How to compute the values?

When values can be calculated with a math formula, it may be more simple to write the formula than all the values. This code:

```
\htablecalc[2]{$x$}{x=-3,-1,1,3,5,7,9,11,13}
              {$2x$}{2*x}
              {$x^2$}{x*x}
```

can be replaced by this shorter oneo:

```
\htablecalc[2]{$x$}{x=a;a=-3:13[2]}
              {$2x$}{2*x}
              {$x^2$}{x*x}
```

The presence of a `";"` changes the analysis of the argument: on the right of `";"` we say that the dummy variable `"a"` varies between -3 and 13 with a step of 2. Therefore is an *odd* integer. On the left of `";"` we say that the dummy variable – here `x` – involved in the formulas used to compute the results is equal to `a` and consequently the values are odd integers between -3 and 13.
These values could have been generated with this argument `{x=2*a+1:a=-2:6}` (the step is 1 by default) or this other one `{x=2*a-3;a=0:8}`, or another one because there are several way to generate a set of values.

When using an argument with `";"`, the feature enabling to hide cells (see page 4) is not available. Moreover, the user should be aware of the number of generated values to avoid huge tables.

With a `";"`, the syntax of the argument is:

⟨*variable 1*⟩`=`⟨*formula*⟩`;`⟨*variable 2*⟩`=`⟨*min*⟩`:`⟨*max*⟩`[`⟨*step*⟩`]`

where:

- ⟨*variable 1*⟩ is the dummy variable involved in the formulas used to compute the results;
- ⟨*variable 2*⟩ is the dummy variable involved in the formulas used to compute the values; it must be different from ⟨*variable 1*⟩;
- ⟨*formula*⟩ is the formula used to compute the values. The variable in this formula is ⟨*variable 2*⟩;
- ⟨*min*⟩`:`⟨*max*⟩ is the interval in which ⟨*variable 2*⟩ varies;

- ⟨*step*⟩ is the step added to ⟨*variable 2*⟩ until it reaches ⟨*max*⟩ or more. It is optional and its defaul value is 1. It must be different from 0.

There are many different ways to generate the same set of values.
For example, the values `{0,1,2,3,4,5,6,7,8,9,10}` can be generated with:

- `{z=x;x=0:10}` and "z" will be the dummy variable in formulas;
- `{n=2*a;a=0:5[0.5]}` and "n" will be the dummy variable in formulas;
- `{x=y/10;y=0:100[10]}` and "x" will be the dummy variable in formulas;

The value of ⟨*step*⟩ and ⟨*min*⟩`:`⟨*max*⟩ must be coherent: `0:10[-1]` will provoke an error message from tabularcalc!

This is an example using the trigonometric functions of `fp`:

```
\htablecalc[6]{$x$\ [deg]}{x=a;a=15:75[15]}
              {$\sin x$}{round(sin(x*pi/180),6)}
              {$\cos x$}{round(cos(x*pi/180),6)}
              {$\tan x$}{round(tan(x*pi/180),6)}
              {$\sin^2x$}{round(sin(x*pi/180)^2,6)}
              {$\cos^2x$}{round(cos(x*pi/180)^2,6)}
              {$\tan^2x$}{round(tan(x*pi/180)^2,6)}
```

$x$ [deg]	15	30	45	60	75
$\sin x$	0.258 819	0.5	0.707 107	0.866 025	0.965 926
$\cos x$	0.965 926	0.866 025	0.707 107	0.5	0.258 819
$\tan x$	0.267 949	0.577 35	1	1.732 051	3.732 051
$\sin^2 x$	0.066 987	0.25	0.5	0.75	0.933 013
$\cos^2 x$	0.933 013	0.75	0.5	0.25	0.066 987
$\tan^2 x$	0.071 797	0.333 333	1	3	13.928 203

And here is another table displaying powers of 10, their decimal logarithm, their square root and their inverse:

```
\htablecalc[3]{Power of 10}{x=round(10^n,4);n=-3:3}
              {Decimal logarithm}{ln(x)/ln(10)}
              {Square root}{round(root(2,x),3)}
              {Inverse}{1/x}
```

Power of 10	0.001	0.01	0.1	1	10	100	1,000
Decimal logarithm	−3	−2	−1	0	1	2	3
Square root	0.032	0.1	0.316	1	3.162	10	31.623
Inverse	1,000	100	10	1	0.1	0.01	0.001

4 Advanced customization

4.1 Put 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:

```
1 \usepackage{color}
2 \defcellcode{2}{3}{\color{blue}}
3 \defcellcode{1}{}{\color{red}}
4 \defcellcode{}{4}{\color{brown}}
5 \htablecalc[3]{$x$}{x=-2,-1,0,1,2,3}
6               {$2x$}{2*x}
7               {$3x$}{3*x}
8               {$4x$}{4*x}
```

x	-2	-1	0	1	2	3
$2x$	-4	-2	0	2	4	6
$3x$	-6	-3	0	3	6	9
$4x$	-8	-4	0	4	8	12

Notice that the cell (1 , 4) whose content is 2 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]{$x$}{x=-2,-1,0,1,2,3}
              {$2x$}{2*x}
              {$3x$}{3*x}
              {$4x$}{4*x}
```

---

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

x	-2	-1	0	1	2	3
$2x$	-4	-2	0	2	4	6
$3x$	-6	-3	0	3	6	9
$4x$	-8	-4	0	4	8	12

4.2 Customizing the number display

4.2.1 Macros `\printvalue` and `\printresult`

To display a value, the macro `\tcprintvalue` 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, `\tcprintvalue` is defined with this code:

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

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

To display a result, the macro `\tcprintresult` 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, `\tcprintresult` is defined with this code:

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

Notice that the argument `#2` (the value) is ignored by `\tcprintresult`. 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 (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\tcprintresult#1#2{%
    \ifdim#1pt<10pt\color{blue}\fi
    \ifdim#2pt<0pt
        \color{red}\texttt{X}%
    \else
        \numprint[cm^2]{#1}%
    \fi}
\htablecalc{length}{x=0.7,-10,3,-2,5,12}
          {Area of square}{x*x}
```

length	0.7	-10	3	-2	5	12
Area of square	0.49 cm^2	`X`	9 cm^2	`X`	25 cm^2	144 cm^2

4.2.2 How to control the rounding of numbers

Results of calculation coming from `fp` have a good precision, and decimal resultas have often many digits. Here is, for example the decimal value of $\sqrt{10}$, computed by `fp`:

$$3.162\,277\,660\,168\,379\,312$$

The first 11 digits are right, the 12th is rounded.

To display results, the `round(number,precision)` of `fp` can be used. To avoid writing many times `round(number,precision)` in the code, tabularcalc provides `\tcprintroundresult`. Its mandatory argument is the precision of the decimal part. The starred macro `\tcprintroundresult*` fills the decimal part with "0" if necessary. If the argument is empty, no rounding is done (default behaviour).

```
\tcprintroundresult{3}
\htablecalc{$x$}{x=2,3,4,5}
           {$\sqrt{x}$}{root(2,x)}
```

x	2	3	4	5
$\sqrt{x}$	1.414	1.732	2	2.236

```
\tcprintroundresult*{3}
\htablecalc{$x$}{x=2,3,4,5}
           {$\sqrt{x}$}{root(2,x)}
```

x	2	3	4	5
$\sqrt{x}$	1.414	1.732	2.000	2.236

To round values, it is not adviced to use the `round` function of `fp` since the rounded values will be used to compute the results, and rounding errors may add up. In this example, the values (square roots of integers) are the are rounded at 10^{-2}, and the results are the square of values:

```
\htablecalc{squre roots}{x=round(root(2,k),2);k=2:4}
           {square}{x*x}
```

squre roots	1.41	1.73	2
square	1.988 1	2.992 9	4

It is obvious that rounding errors are taken into account to compute results.
It is better to use `\tcprintroundvalue` which works like `\tcprintroundresult`:

```
\tcprintroundvalue{2}
\htablecalc{squre roots}{x=root(2,k);k=2:4}
           {square}{x*x}
```

squre roots	1.41	1.73	2
square	1.999 999 999 999 999 98	2.999 999 999 999 999 935	3.999 999 999 999 999 96

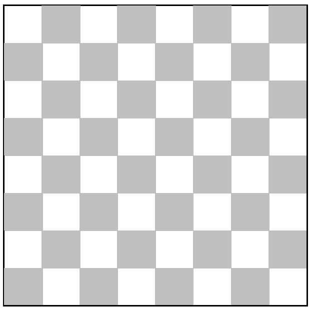
The results – which are not rounded – are much nearer the expected integers.

4.2.3 For the fun

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

- on line 1, 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 2;
- horizontal lines of the top and bottom of the table are drawn (line 3), and vertical lines of the left and right (line 4);
- a strut 0.5 cm height is defined to be displayed at the begining of every row (line 5);
- finally, if the sum of the row number and the column number is odd, the square is filled of gray (line 7 and 8).

```
1  \arraycolsep=0pt\tabcolsep=0pt
2  \def\tcprintvalue#1{}\def\tcprintresult#1#2{}
3  \tcsethrule{\hline}{}{}
4  \tcsetcoltype[|]{|m{0.5cm}}{m{0.5cm}}
5  \def\tcatbeginrow{\rule[-0.2cm]{0pt}{0.3cm}}
6  \edefcellcode{}{}{%
7      \ifodd\numexpr\tccol+\tclin\relax
8          \noexpand\cellcolor{lightgray}\fi
9  }
10 \htablecalc[7]{}{x=1,2,3,4,5,6,7}
11     {}{x}{}{x}{}{x}{}{x}{}{x}{}{x}{}{x}
```



5 Export a table in a file

No matter how customizable tabularcalc is, some tables need fine adjustments by the user at the keyboard. The `\tcwritetofile{⟨filename⟩}` has a mandatory argument which is the name of a file without extension. The next `\htablecalc` or `\vtablecalc` after this command will not display the tables, but a file named ⟨*filename*⟩`.tex` will be written in the current directory, and its content will be the code of the table.

Here is an example:

```
\tcwritetofile{mytable}
\defcellcode{}{2}{\color{blue}}
\htablecalc[2]{$x$}{x=k;k=0:4}
               {$2x$}{2*x}
               {$x^2$}{x*x}
\tcresetcellcode
```

A file "`mytable.tex`" is created in the current directory and its content is the code of the table:

```
\begin {tabular}{|c|*{5}{c|}}\hline
\tcatbeginrow $x$&\tcprintvalue {0}&\color {blue}\tcprintvalue {1}&\tcprintvalue {2}&\tcprintvalue {3}&\tcprintvalue {4}\\\hline
\tcatbeginrow $2x$&\tcprintresult {0}{0}&\color {blue}\tcprintresult {2}{1}&\tcprintresult {4}{2}&\tcprintresult {6}{3}&\tcprintresult {8}{4}\\ \hline
\tcatbeginrow $x^2$&\tcprintresult {0}{0}&\color {blue}\tcprintresult {1}{1}&\tcprintresult {4}{2}&\tcprintresult {9}{3}&\tcprintresult {16}{4}\\ \hline
\end {tabular}
```

The user can modify this code, and then this file can be included in the LaTeX code with:

```
\input{mytable.tex}
```

and here is the result:

x	0	1	2	3	4
$2x$	0	2	4	6	8
x^2	0	1	4	9	16

6 Use infix or postfix notation

Infix or postfix notation can be used since `\FPeval` accepts both. In this exaple, the same table is generated with each notation. The result is exactly the same since only notation changes while math engine is the same:

```
\tcprintroundvalue{6}
\tcprintroundresult{6}
With infix notation\par
\htablecalc[3]{$x=10^k$ o\`u $k\in[-3;3]$}{x=10^k;k=-3:3}
               {$\log x$}{ln(x)/ln(10)}
               {$\sqrt{x}$}{root(2,x)}
               {$\frac{1}{x}$}{1/x}

\medskip
With postfix notation\par
\htablecalc[3]{$x=10^k$ o\`u $k\in[-3;3]$}{x=k 10 pow;k=-3:3}
               {$\log x$}{x ln 10 ln div}
               {$\sqrt{x}$}{2 x root}
               {$\frac{1}{x}$}{1 x div}
```

With infix notation

$x = 10^k$ où $k \in [-3; 3]$	0.001	0.01	0.1	1	10	100	1,000
$\log x$	−3	−2	−1	0	1	2	3
$\sqrt{x}$	0.031 623	0.1	0.316 228	1	3.162 278	10	31.622 777
$\frac{1}{x}$	1,000	100	10	1	0.1	0.01	0.001

With postfix notation

$x = 10^k$ où $k \in [-3; 3]$	0.001	0.01	0.1	1	10	100	1,000
$\log x$	−3	−2	−1	0	1	2	3
$\sqrt{x}$	0.031 623	0.1	0.316 228	1	3.162 278	10	31.622 777
$\frac{1}{x}$	1,000	100	10	1	0.1	0.01	0.001

If possible and if the user is used to it, the postfix notation should be prefered beacuse it often saves computation times. Indeed, to compute $\cos x(1 - \cos x)$, here is the infix notation:

```
cos(x)*(1-cos(x))
```

Obviously, $\cos x$ is unnecessarily computed twice which slows down the compilation.

With the postfix notation, it is computed once:

```
x cos copy 1 swap sub mul
```

* * *

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