Manual for Package PgfplotsTable

Component of pgfplots, Version 1.1

http://sourceforge.net/projects/pgfplots

Christian Feuersänger*
Institut für Numerische Simulation
Universität Bonn, Germany

August 4, 2008

Abstract

This package reads tab-separated numerical tables from input and generates code for pretty-printed LaTeX-tabulars. It rounds to the desired precision and prints it in different number formatting styles.

Contents

1 Introduction

PgfplotsTable is a lightweight sub-package of pgfplots which employs its table input methods and the number formatting techniques to convert tab-separated tables into tabulars.

Its input is a text file containing space separated rows, possibly starting with column names. Its output is a LaTeX tabular[1] which contains selected columns of the text table, rounded to the desired precision, printed in the desired number format (fixed point, integer, scientific etc).

*`http://wissrech.ins.uni-bonn.de/people/feuersaenger`

[1] Please see the remarks in section 10 for plain TeX and ConTeXt.

It is used with

```
\usepackage{pgfplotstable}
% recommended:
%\usepackage{booktabs}
%\usepackage{array}
%\usepackage{colortbl}
```

and requires PGFPLOTS and PGF $\geq$ 2.00 installed.

Knowledge of `pgfkeys` is useful for a deeper insight into this package. Please refer to the PGF manual, [1, section pgfkeys] if you want a deeper insight into `pgfkeys`. Otherwise, simply skip over to the examples provided in this document.

2 Text table input format

PGFPLOTSTABLE works with plain text file tables in which entries ("cells") are separated by a separation character. The initial separation character is "white space" which means "at least one space or tab". Those tables can have a header line which contains column names and most other columns typically contain numerical data.
The following listing shows `pgfplotstable.example1.dat` and is used often throughout this documentation.

```
# Convergence results
# fictional source, generated 2008
level    dof      error1            error2   info     grad(log(dof),log(error2)) quot(error1)
1        4        2.50000000e-01    7.57858283e-01    48       0                    0
2        16       6.25000000e-02    5.00000000e-01    25       -3.00000000e-01   4
3        64       1.56250000e-02    2.87174589e-01    41       -3.99999999e-01   4
4        256      3.90625000e-03    1.43587294e-01    8        -5.00000003e-01   4
5        1024     9.76562500e-04    4.41941738e-02    22       -8.49999999e-01   4
6        4096     2.44140625e-04    1.69802322e-02    46       -6.90000001e-01   4
7        16384    6.10351562e-05    8.20091159e-03    40       -5.24999999e-01   4
8        65536    1.52587891e-05    3.90625000e-03    48       -5.35000000e-01   3.99999999e+00
9        262144   3.81469727e-06    1.95312500e-03    33       -5.00000000e-01   4.00000001e+00
10       1048576  9.53674316e-07    9.76562500e-04    2        -5.00000000e-01   4.00000001e+00
```

Lines starting with '`%`' or '`#`' are considered to be comment lines and are ignored.

There is future support for a second header line which must start with '`$flags `' (the space is obligatory, even if the column separator is *not* space!). Currently, such a line is ignored. It may be used to provide number formatting options like precision and number format.

`/pgfplots/table/col sep=space|comma|semicolon|colon|braces` (no default, initially `space`)

Specifies the column separation character for table reading. The initial choice, `space` means "at least one white space". White spaces are tab stops or spaces (newlines always delimit lines).

For example, the file `pgfplotstable.example1.csv` uses commas as separation characters.

```
# Convergence results
# fictional source  generated 2008
level,dof,error1,error2,info,{grad(log(dof),log(error2))},quot(error1)
1,9,2.50000000e-01,7.57858283e-01,48,0,0
2,25,6.25000000e-02,5.00000000e-01,25,-1.35691545e+00,4
3,81,1.56250000e-02,2.87174589e-01,41,-1.17924958e+00,4
4,289,3.90625000e-03,1.43587294e-01,8,-1.08987331e+00,4
5,1089,9.76562500e-04,4.41941738e-02,22,-1.04500712e+00,4
6,4225,2.44140625e-04,1.69802322e-02,46,-1.02252239e+00,4
7,16641,6.10351562e-05,8.20091159e-03,40,-1.01126607e+00,4
8,66049,1.52587891e-05,3.90625000e-03,48,-1.00563427e+00,3.99999999e+00
9,263169,3.81469727e-06,1.95312500e-03,33,-1.00281745e+00,4.00000001e+00
10,1050625,9.53674316e-07,9.76562500e-04,2,-1.00140880e+00,4.00000001e+00
```

Thus, we need to specify `col sep=comma` when we read it.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	$6.25 \cdot 10^{-2}$	0.5	25	−0.3	4
3	64	$1.56 \cdot 10^{-2}$	0.29	41	−0.4	4
4	256	$3.91 \cdot 10^{-3}$	0.14	8	−0.5	4
5	1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	22	−0.85	4
6	4,096	$2.44 \cdot 10^{-4}$	$1.7 \cdot 10^{-2}$	46	−0.69	4
7	16,384	$6.1 \cdot 10^{-5}$	$8.2 \cdot 10^{-3}$	40	−0.52	4
8	65,536	$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	48	−0.54	4
9	$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	33	−0.5	4
10	$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	−0.5	4

```
\pgfplotstabletypesetfile[col sep=comma]{pgfplotstable.example1.csv}
```

You may call `\pgfplotstableset{col sep=comma}` once in your preamble if all your tables use commas as column separator.

Please note that if cell entries (for example column names) contain the separation character, you need to enclose the column entry in *braces*: `{grad(log(dof),log(error2)}`. If you want to use unmatched braces, you need to write a backslash before the brace. For example the name '`column{withbrace`' needs to be written as '`column\{withbrace`'.

Furthermore, if you need empty cells in case `col sep=space`, you have to provide `{}` to delimit such a cell since `col sep=space` uses *at least* one white space (consuming all following ones).

The value `col sep=braces` is special since it actually uses two separation characters. Every single cell entry is delimited by an opening and a closing brace, `{`⟨*entry*⟩`}`, for this choice. Furthermore, any wite spaces (spaces and tabs) between cell entries are *skipped* in case `braces` until the next `{`⟨*entry*⟩`}` is found.

`\pgfplotstabletypesetfile[`⟨*optional arguments*⟩`]{`⟨*file name*⟩`}`

This command loads the table file `{`⟨*file name*⟩`}` and typesets it using the current configuration of number formats and table options.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	$6.25 \cdot 10^{-2}$	0.5	25	−0.3	4
3	64	$1.56 \cdot 10^{-2}$	0.29	41	−0.4	4
4	256	$3.91 \cdot 10^{-3}$	0.14	8	−0.5	4
5	1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	22	−0.85	4
6	4,096	$2.44 \cdot 10^{-4}$	$1.7 \cdot 10^{-2}$	46	−0.69	4
7	16,384	$6.1 \cdot 10^{-5}$	$8.2 \cdot 10^{-3}$	40	−0.52	4
8	65,536	$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	48	−0.54	4
9	$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	33	−0.5	4
10	$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	−0.5	4

```
\pgfplotstabletypesetfile{pgfplotstable.example1.dat}
```

The configuration can be customized with ⟨*optional arguments*⟩.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	6.25_{-2}	0.5	25	−0.3	4
3	64	1.56_{-2}	0.29	41	−0.4	4
4	256	3.91_{-3}	0.14	8	−0.5	4
5	1,024	9.77_{-4}	4.42_{-2}	22	−0.85	4
6	4,096	2.44_{-4}	1.70_{-2}	46	−0.69	4
7	16,384	6.10_{-5}	8.20_{-3}	40	−0.52	4
8	65,536	1.53_{-5}	3.91_{-3}	48	−0.54	4
9	2.62_{5}	3.81_{-6}	1.95_{-3}	33	−0.5	4
10	1.05_{6}	9.54_{-7}	9.77_{-4}	2	−0.5	4

```
\pgfplotstabletypesetfile[sci subscript,sci zerofill]{pgfplotstable.example1.dat}
```

`\pgfplotstableread{`⟨*file name*⟩`}{`⟨*\macro*⟩`}`

Loads the table `{`⟨*file name*⟩`}` into a TEX-macro ⟨*\macro*⟩. This macro can than be used several times.

dof	error1	dof	error2
4	0.25	4	0.76
16	$6.25 \cdot 10^{-2}$	16	0.5
64	$1.56 \cdot 10^{-2}$	64	0.29
256	$3.91 \cdot 10^{-3}$	256	0.14
1,024	$9.77 \cdot 10^{-4}$	1,024	$4.42 \cdot 10^{-2}$
4,096	$2.44 \cdot 10^{-4}$	4,096	$1.7 \cdot 10^{-2}$
16,384	$6.1 \cdot 10^{-5}$	16,384	$8.2 \cdot 10^{-3}$
65,536	$1.53 \cdot 10^{-5}$	65,536	$3.91 \cdot 10^{-3}$
$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	$2.62 \cdot 10^{5}$	$1.95 \cdot 10^{-3}$
$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$1.05 \cdot 10^{6}$	$9.77 \cdot 10^{-4}$

```
\pgfplotstableread{pgfplotstable.example1.dat}\table
\pgfplotstabletypeset[columns={dof,error1}]\table
\hspace{2cm}
\pgfplotstabletypeset[columns={dof,error2}]\table
```

`\pgfplotstabletypeset[`⟨*optional arguments*⟩`]{`⟨*\macro*⟩`}`

Works in the same way as `\pgfplotstabletypesetfile`, but it accepts an already loaded table.

`/pgfplots/table/header={`⟨*boolean*⟩`}` (no default, initially `true`)

Configures if column names shall be identified during input operations.

The 'header' is the first non-comment line, but only if this line contains non-numerical entries. If the first non-comment line contains at least one non-numerical entry (for example '`a name`'), each entry in this line is supposed to be a column name.

If the first non-comment line contains only numerical data, it is used as data row. In this case, column indices will be assigned as column "names".

Use `header=false` to force this last case, i.e. even if the first line contains strings, they won't be recognised as column names.

3 Selecting columns

`/pgfplots/table/columns={`⟨*comma-separated-list*⟩`}` (no default, initially `all available ones`)

Selects particular columns the table. If this option is missing, all available columns will be selected.

Inside of `{`⟨*comma-separated-list*⟩`}`, column names as they appear in the table's header are expected. If there is no header, simply use column names. If there are column names, the special syntax `[index]`⟨*integer*⟩ can be used to select columns by index. The first column has index 0.

dof	level	info
4	1	48
16	2	25
64	3	41
256	4	8
1,024	5	22
4,096	6	46
16,384	7	40
65,536	8	48
$2.62 \cdot 10^{5}$	9	33
$1.05 \cdot 10^{6}$	10	2

```
\pgfplotstabletypesetfile[columns={dof,level,[index]4}]{pgfplotstable.example1.dat}
```

`/pgfplots/table/alias/`⟨*col name*⟩`/.initial={`⟨*real col name*⟩`}`

Assigns the new name ⟨*new col name*⟩ for the column denoted by ⟨*real col name*⟩. Afterwards, accessing ⟨*new col name*⟩ will use the data associated with column ⟨*real col name*⟩.

You can use `columns/⟨new col name⟩/.style` to assign styles for the alias, not for the original column name.

If there exists both an alias and a column of the same name, the column name will be preferred. Furthermore, if there exists a `create on use` statement with the same name, this one will also be preferred.

In case ⟨*new col name*⟩ contains characters which are required for key settings, you need to use braces around it: "`alias/{name=wi/th,special}/.initial={othername}`".

/pgfplots/table/columns/⟨*column name*⟩/.style={⟨*key-value-list*⟩}

Sets all options in {⟨*key-value-list*⟩} exclusively for {⟨*column name*⟩}.

level	Dof	L_2	A	info	grad(log(dof),log(error2))	quot(error1)
1	4	2.500_{-1}	7.58_{-1}	48	0	0
2	16	6.250_{-2}	5.00_{-1}	25	−0.3	4
3	64	1.563_{-2}	2.87_{-1}	41	−0.4	4
4	256	3.906_{-3}	1.44_{-1}	8	−0.5	4
5	1,024	9.766_{-4}	4.42_{-2}	22	−0.85	4
6	4,096	2.441_{-4}	1.70_{-2}	46	−0.69	4
7	16,384	6.104_{-5}	8.20_{-3}	40	−0.52	4
8	65,536	1.526_{-5}	3.91_{-3}	48	−0.54	4
9	262,144	3.815_{-6}	1.95_{-3}	33	−0.5	4
10	1,048,576	9.537_{-7}	9.77_{-4}	2	−0.5	4

```
\pgfplotstabletypesetfile[
    columns/error1/.style={
        column name=$L_2$,
        sci,sci zerofill,sci subscript,
        precision=3},
    columns/error2/.style={
        column name=$A$,
        sci,sci zerofill,sci subscript,
        precision=2},
    columns/dof/.style={
        int detect,
        column name=\textsc{Dof}
    }
]
    {pgfplotstable.example1.dat}
```

If your column name contains commas '`,`', slashes '`/`' or equal signs '`=`', you need to enclose the column name in braces.

Dof	L_2	slopes L_2
4	2.500_{-1}	0.0
16	6.250_{-2}	−0.3
64	1.563_{-2}	−0.4
256	3.906_{-3}	−0.5
1,024	9.766_{-4}	−0.8
4,096	2.441_{-4}	−0.7
16,384	6.104_{-5}	−0.5
65,536	1.526_{-5}	−0.5
262,144	3.815_{-6}	−0.5
1,048,576	9.537_{-7}	−0.5

```
\pgfplotstabletypesetfile[
    columns={dof,error1,{grad(log(dof),log(error2))}},
    columns/error1/.style={
        column name=$L_2$,
        sci,sci zerofill,sci subscript,
        precision=3},
    columns/dof/.style={
        int detect,
        column name=\textsc{Dof}},
    columns/{grad(log(dof),log(error2))}/.style={
        column name=slopes $L_2$,
        fixed,fixed zerofill,
        precision=1}
]
    {pgfplotstable.example1.dat}
```

If your tables don't have column names, you can simply use integer indices instead of {⟨*column name*⟩} to refer to columns. If you have column names, you can't set column styles using indices.

/pgfplots/table/display columns/⟨*index*⟩/.style={⟨*key-value-list*⟩}

Applies all options in {⟨*key-value-list*⟩} exclusively to the column which will appear at position ⟨*index*⟩ in the output table.

In contrast to the `table/columns/`⟨*name*⟩ styles, this option refers to the output table instead of the input table. Since the output table has no unique column name, you can only access columns by index.

Indexing starts with ⟨*index*⟩ = 0.

Display column styles override input column styles.

/pgfplots/table/column type={⟨*tabular column type*⟩} (no default, initially c)

Contains the column type for `tabular`.

If all column types are empty, the complete argument is skipped (assuming that no `tabular` environment is generated).

Use `\pgfplotstableset{column type/.add={⟨before⟩}{⟨after⟩}}` to *modify* a value instead of overwriting it. The `.add` key handler works for other options as well.

dof	error1	info
4	0.25	48
16	$6.25 \cdot 10^{-2}$	25
64	$1.56 \cdot 10^{-2}$	41
256	$3.91 \cdot 10^{-3}$	8
1,024	$9.77 \cdot 10^{-4}$	22
4,096	$2.44 \cdot 10^{-4}$	46
16,384	$6.1 \cdot 10^{-5}$	40
65,536	$1.53 \cdot 10^{-5}$	48
$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	33
$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	2

```
\pgfplotstabletypesetfile[
    columns={dof,error1,info},
    column type/.add={|}{}% results in 'c|'
]
    {pgfplotstable.example1.dat}
```

/pgfplots/table/assign column name/.code={⟨...⟩}

Allows to *modify* the value of `column name`.

Argument `#1` is the current column name, that means after evaluation of `column name`. After `assign column` name, a new (possibly modified) value for `column name` should be set.

That means you can use `column name` to assign the name as such and `assign column name` to generate final TEX code (for example to insert `\multicolumn{1}{c}{#1}`).

Default is empty which means no change.

/pgfplots/table/multicolumn names={⟨*tabular column type*⟩} (style, no default, initially c)

A style which typesets each column name using a `\multicolumn{1}{`⟨*tabular column type*⟩`}{`⟨*the column name*⟩`}` statement.

/pgfplots/table/dec sep align={⟨*header column type*⟩} (style, no default, initially c)

A style which aligns numerical columns at the decimal separator.

The first argument determines the alignment of the header column.

The style `dec sep align` actually introduces two new `tabular` columns, namely `r@{}l`. It introduces multicolumns for column names accordingly and handles numbers which do not have a decimal separator.

Please note that you need `\usepackage{array}` for this style.

dof	error1	error2	info	grad(log(dof),log(error2))
4	0.25	7.58_{-1}	48	0
16	$6.25 \cdot 10^{-2}$	5.00_{-1}	25	−0.3
64	$1.56 \cdot 10^{-2}$	2.87_{-1}	41	−0.4
256	$3.91 \cdot 10^{-3}$	1.44_{-1}	8	−0.5
1,024	$9.77 \cdot 10^{-4}$	4.42_{-2}	22	−0.85
4,096	$2.44 \cdot 10^{-4}$	1.70_{-2}	46	−0.69
16,384	$6.1 \cdot 10^{-5}$	8.20_{-3}	40	−0.52
65,536	$1.53 \cdot 10^{-5}$	3.91_{-3}	48	−0.54
$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	1.95_{-3}	33	−0.5
$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	9.77_{-4}	2	−0.5

```
% requires \usepackage{array}
\pgfplotstabletypesetfile[
    columns={dof,error1,error2,info,{grad(log(dof),log(error2))}},
    columns/error1/.style={dec sep align},
    columns/error2/.style={sci,sci subscript,sci zerofill,dec sep align},
    columns/info/.style={fixed,dec sep align},
    columns/{grad(log(dof),log(error2))}/.style={fixed,dec sep align}
]
    {pgfplotstable.example1.dat}
```

Or with comma as decimal separator:

dof	error1	error2	info	grad(log(dof),log(error2))
4	0,25	$7{,}58_{-1}$	48	0
16	$6{,}25 \cdot 10^{-2}$	$5{,}00_{-1}$	25	−0,3
64	$1{,}56 \cdot 10^{-2}$	$2{,}87_{-1}$	41	−0,4
256	$3{,}91 \cdot 10^{-3}$	$1{,}44_{-1}$	8	−0,5
1.024	$9{,}77 \cdot 10^{-4}$	$4{,}42_{-2}$	22	−0,85
4.096	$2{,}44 \cdot 10^{-4}$	$1{,}70_{-2}$	46	−0,69
16.384	$6{,}1 \cdot 10^{-5}$	$8{,}20_{-3}$	40	−0,52
65.536	$1{,}53 \cdot 10^{-5}$	$3{,}91_{-3}$	48	−0,54
$2{,}62 \cdot 10^{5}$	$3{,}81 \cdot 10^{-6}$	$1{,}95_{-3}$	33	−0,5
$1{,}05 \cdot 10^{6}$	$9{,}54 \cdot 10^{-7}$	$9{,}77_{-4}$	2	−0,5

```
% requires \usepackage{array}
\pgfplotstabletypesetfile[
    use comma,
    columns={dof,error1,error2,info,{grad(log(dof),log(error2))}},
    columns/error1/.style={dec sep align},
    columns/error2/.style={sci,sci subscript,sci zerofill,dec sep align},
    columns/info/.style={fixed,dec sep align},
    columns/{grad(log(dof),log(error2))}/.style={fixed,dec sep align}
]
    {pgfplotstable.example1.dat}
```

It may be advisable to use the `zerofill` variants to force at least one digit after the decimal separator.

Please note that this style overwrites `column type`, `assign cell content` and some number formatting settings.

`/pgfplots/table/sci sep align={`⟨*header column type*⟩`}` (style, no default, initially `c`)

A style which aligns numerical columns at the exponent in scientific representation.

The first argument determines the alignment of the header column.

It works similiarly to `dec sep align`, namely by introducing two artificial columns `r@{}l` for alignment.

Please note that you need `\usepackage{array}` for this style.

Please note that this style overwrites `column type`, `assign cell content` and some number formatting settings.

`/pgfplots/table/dcolumn={`⟨*tabular column type*⟩`}{`⟨*type for column name*⟩`}` (style, no default, initially `{D{.}{.}{2}}{c}`)

A style which can be used together with the `dcolumn` package of David Carlisle. It also enables alignment at the decimal separator. However, the decimal separator needs to be exactly one character which is incompatible with '`{,}`' (the default setting for `use comma`).

`/pgfplots/table/every first column` (style, no value)

A style which is installed for every first column only.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	$6.25 \cdot 10^{-2}$	0.5	25	−0.3	4
3	64	$1.56 \cdot 10^{-2}$	0.29	41	−0.4	4
4	256	$3.91 \cdot 10^{-3}$	0.14	8	−0.5	4
5	1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	22	−0.85	4
6	4,096	$2.44 \cdot 10^{-4}$	$1.7 \cdot 10^{-2}$	46	−0.69	4
7	16,384	$6.1 \cdot 10^{-5}$	$8.2 \cdot 10^{-3}$	40	−0.52	4
8	65,536	$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	48	−0.54	4
9	$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	33	−0.5	4
10	$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	−0.5	4

```
\pgfplotstabletypesetfile[
 every head row/.style={before row=\hline,after row=\hline\hline},
 every last row/.style={after row=\hline},
 every first column/.style={
  column type/.add={|}{}
 },
 every last column/.style={
  column type/.add={}{|}
 }]
    {pgfplotstable.example1.dat}
```

/pgfplots/table/every last column (style, no value)

A style which is installed for every last column only.

/pgfplots/table/every even column (style, no value)

A style which is installed for every column with even column index (starting with 0).

```
\pgfplotstableset{
    columns={dof,error1,{grad(log(dof),log(error2))},info},
    columns/error1/.style={
        column name=$L_2$,
        sci,sci zerofill,sci subscript,
        precision=3},
    columns/dof/.style={
        int detect,
        column name=\textsc{Dof}},
    columns/{grad(log(dof),log(error2))}/.style={
        column name=slopes $L_2$,
        fixed,fixed zerofill,
        precision=1}}
```

Dof	L_2	slopes L_2	info
4	2.500_{-1}	0.0	48
16	6.250_{-2}	−0.3	25
64	1.563_{-2}	−0.4	41
256	3.906_{-3}	−0.5	8
1,024	9.766_{-4}	−0.8	22
4,096	2.441_{-4}	−0.7	46
16,384	6.104_{-5}	−0.5	40
65,536	1.526_{-5}	−0.5	48
262,144	3.815_{-6}	−0.5	33
1,048,576	9.537_{-7}	−0.5	2

```
% requires \usepackage{colortbl}
\pgfplotstabletypesetfile[
 every even column/.style={
  column type/.add={>{\columncolor[gray]{.8}}}{}
}]
    {pgfplotstable.example1.dat}
```

/pgfplots/table/every odd column (style, no value)

A style which is installed for every column with odd column index (starting with 0).

\pgfplotstablecol

During the evaluation of row or column options, this command expands to the current columns' index.

\pgfplotstablerow

During the evaluation of row or column options, this command expands to the current rows' index.

\pgfplotstablecols

During the evaluation of row or column options, this command expands to the total number of columns in the output table.

\pgfplotstablerows

During evaluation of *columns*, this command expands to the total number of *input* rows. You can use it inside of `row predicate`.

During evaluation of *rows*, this command expands to the total number of *output* rows.

4 Configuring row appearance

The following styles allow to configure the final table code *after any cell contents have been assigned.*

/pgfplots/table/before row={⟨*TEX code*⟩} (no default)

Contains TEX code which will be installed before the first cell in a row.

/pgfplots/table/after row={⟨*TEX code*⟩} (no default)

Contains TEX code which will be installed after the last cell in a row (i.e. after \\).

/pgfplots/table/every even row (style, no value)

A style which is installed for each row with even row index. The first row is supposed to be a "head" row and does not count. Indexing starts with 0.

```
\pgfplotstableset{
    columns={dof,error1,{grad(log(dof),log(error2))}},
    columns/error1/.style={
        column name=$L_2$,
        sci,sci zerofill,sci subscript,
        precision=3},
    columns/dof/.style={
        int detect,
        column name=\textsc{Dof}},
    columns/{grad(log(dof),log(error2))}/.style={
        column name=slopes $L_2$,
        fixed,fixed zerofill,
        precision=1}}
```

Dof	L_2	slopes L_2
4	2.500_{-1}	0.0
16	6.250_{-2}	−0.3
64	1.563_{-2}	−0.4
256	3.906_{-3}	−0.5
1,024	9.766_{-4}	−0.8
4,096	2.441_{-4}	−0.7
16,384	6.104_{-5}	−0.5
65,536	1.526_{-5}	−0.5
262,144	3.815_{-6}	−0.5
1,048,576	9.537_{-7}	−0.5

```
% requires \usepackage{booktabs}
\pgfplotstabletypesetfile[
    every head row/.style={
        before row=\toprule,after row=\midrule},
    every last row/.style={
        after row=\bottomrule},
]
    {pgfplotstable.example1.dat}
```

Dof	L_2	slopes L_2
4	2.500_{-1}	0.0
16	6.250_{-2}	−0.3
64	1.563_{-2}	−0.4
256	3.906_{-3}	−0.5
1,024	9.766_{-4}	−0.8
4,096	2.441_{-4}	−0.7
16,384	6.104_{-5}	−0.5
65,536	1.526_{-5}	−0.5
262,144	3.815_{-6}	−0.5
1,048,576	9.537_{-7}	−0.5

```
% requires \usepackage{booktabs,colortbl}
\pgfplotstabletypesetfile[
    every even row/.style={
        before row={\rowcolor[gray]{0.9}}},
    every head row/.style={
        before row=\toprule,after row=\midrule},
    every last row/.style={
        after row=\bottomrule},
]
    {pgfplotstable.example1.dat}
```

/pgfplots/table/every odd row (style, no value)

A style which is installed for each row with odd row index. The first row is supposed to be a "head" row and does not count. Indexing starts with 0.

/pgfplots/table/every head row (style, no value)

A style which is installed for each first row in the tabular. This can be used to adjust options for column names or to add extra lines/colours.

/pgfplots/table/every first row (style, no value)

A style which is installed for each first *data* row, i.e. after the head row.

/pgfplots/table/every last row (style, no value)

A style which is installed for each last *data* row.

5 Determining cell contents

The conversion from an unprocessed input table to a final typesetted `tabular` code uses three stages for every cell,

1. Preprocessing,
2. Typesetting,
3. Postprocessing.

The main idea is to select one typesetting algorithm (for example "format my numbers with the configured number style"). This algorithm usually doesn't need to be changed. Fine tuning can then be done using zero, one or more preprocessors and postprocessors. Preprocessing can mean to select only particular rows or to apply some sort of operation before the typesetting algorithm sees the content. Postprocessing means to apply fine-tuning to the resulting TEX output – for example to deal with empty cells or to insert unit suffixes or modify fonts for single cells.

5.1 Typesetting cells

/pgfplots/table/assign cell content/.code={⟨...⟩}

Allows to redefine the algorithm which assigns cell contents. The argument `#1` is the (unformatted) contents of the input table.

The resulting output needs to be written to `/pgfplots/table/@cell content`.

Please note that you may need special attention for `#1={⟨⟩}`, i.e. the empty string. This may happen if a column has less rows than the first column. PgfplotsTable will balance columns automatically in this case, inserting enough empty cells to match the number of rows of the first column.

Please note further that if any column has more entries than the first column, these entries will be skipped and a warning message will be issued into the log file.

This key is evaluated inside of a local TEX group, so any local macro assignments will be cleared afterwards.

/pgfplots/table/assign cell content as number (no value)

This here is the default implementation of `assign cell contents`.

It invokes `\pgfmathprintnumberto` and writes the result into `@cell content`.

/pgfplots/table/string type (style, no value)

A style which redefines `assign cell contents` to simply return the "raw" input data, that means as text column. This assumes input tables with valid LaTeX content (verbatim printing is not supported).

5.2 Preprocessing cells

/pgfplots/table/preproc cell content/.code={⟨...⟩}

Allows to *modify* the contents of cells *before* `assign cell contents` is called.

The semantics is as follows: before the preprocessor, `@cell content` contains the raw input data (or, maybe, the result of another preprocessor call). After the preprocessor, `@cell content` is filled with a – possibly modified – value. The resulting value is then used as input to `assign cell contents`.

In the default settings, `assign cell contents` expects numerical input. So, the preprocessor is expected to produce numerical output.

It is possible to provide multiple preprocessor directives using `.append code` or `.append style` key handlers.

In case you don't want (or need) stackable preprocessors, you can also use '`#1`' to get the raw input datum as it is found in the file.

/pgfplots/table/multiply -1 (style, no value)

Appends code to current `preproc cell content` value which multiplies every cell with -1.

dof	error2	slopes2	dof	error2	slopes2
4	$7.58 \cdot 10^{-1}$	–	4	$7.58 \cdot 10^{-1}$	–
16	$5.00 \cdot 10^{-1}$	-0.3	16	$5.00 \cdot 10^{-1}$	0.3
64	$2.87 \cdot 10^{-1}$	-0.4	64	$2.87 \cdot 10^{-1}$	0.4
256	$1.44 \cdot 10^{-1}$	-0.5	256	$1.44 \cdot 10^{-1}$	0.5
1,024	$4.42 \cdot 10^{-2}$	-0.85	1,024	$4.42 \cdot 10^{-2}$	0.85
4,096	$1.70 \cdot 10^{-2}$	-0.69	4,096	$1.70 \cdot 10^{-2}$	0.69
16,384	$8.20 \cdot 10^{-3}$	-0.53	16,384	$8.20 \cdot 10^{-3}$	0.53
65,536	$3.91 \cdot 10^{-3}$	-0.53	65,536	$3.91 \cdot 10^{-3}$	0.53
$2.62 \cdot 10^{5}$	$1.95 \cdot 10^{-3}$	-0.5	$2.62 \cdot 10^{5}$	$1.95 \cdot 10^{-3}$	0.5
$1.05 \cdot 10^{6}$	$9.77 \cdot 10^{-4}$	-0.5	$1.05 \cdot 10^{6}$	$9.77 \cdot 10^{-4}$	0.5

```
\pgfplotstableset{
    columns={dof,error2,slopes2},
    columns/error2/.style={sci,sci zerofill},
    columns/slopes2/.style={dec sep align,empty cells with={\ensuremath{-}}},
    create on use/slopes2/.style=
        {create col/gradient loglog={dof}{error2}}}

\pgfplotstabletypesetfile{pgfplotstable.example1.dat}

\pgfplotstabletypesetfile[columns/slopes2/.append style={multiply -1}]
    {pgfplotstable.example1.dat}
```

/pgfplots/table/row predicate/.code={⟨...⟩}

A boolean predicate which allows to select particular rows of the input table. The argument `#1` contains the current row's index (starting with 0, not counting comment lines or column names).

The return value is assigned to the TeX-if `\ifpgfplotstableuserow`. If the boolean is not changed, the return value is true.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	$6.25 \cdot 10^{-2}$	0.5	25	−0.3	4
3	64	$1.56 \cdot 10^{-2}$	0.29	41	−0.4	4
4	256	$3.91 \cdot 10^{-3}$	0.14	8	−0.5	4
5	1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	22	−0.85	4
9	$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	33	−0.5	4
10	$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	−0.5	4

```
% requires \usepackage{booktabs}
\pgfplotstabletypesetfile[
    every head row/.style={
        before row=\toprule,after row=\midrule},
    every last row/.style={
        after row=\bottomrule},
    row predicate/.code={%
        \ifnum#1>4\relax
            \ifnum#1<8\relax
                \pgfplotstableuserowfalse
            \fi
        \fi}
]
    {pgfplotstable.example1.dat}
```

Please note that `row predicate` is applied *before* any other option which affects row appearance. It is evaluated before `assign cell contents`. For example, the even/odd row styles refer to those rows for which the predicate returns `true`. In fact, you can use `row predicate` to truncate the complete table before it as actually processed.

During `row predicate`, the macro `\pgfplotstablerows` contains the total number of *input* rows.

Furthermore, `row predicate` applies only to the typeset routines, not the read methods. If you want to plot only selected table entries with `\addplot table`, use the PGFPLOTS coordinate filter options.

`/pgfplots/table/skip rows between index={`⟨*begin*⟩`}{`⟨*end*⟩`}` (style, no default)

A style which appends an `row predicate` which discards selected rows. The selection is done by index where indexing starts with 0. Every row with index ⟨*begin*⟩ ≤ *i* < ⟨*end*⟩ will be skipped.

level	dof	error1	error2	info	grad(log(dof),log(error2))	quot(error1)
1	4	0.25	0.76	48	0	0
2	16	$6.25 \cdot 10^{-2}$	0.5	25	−0.3	4
5	1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	22	−0.85	4
6	4,096	$2.44 \cdot 10^{-4}$	$1.7 \cdot 10^{-2}$	46	−0.69	4
7	16,384	$6.1 \cdot 10^{-5}$	$8.2 \cdot 10^{-3}$	40	−0.52	4
10	$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	−0.5	4

```
% requires \usepackage{booktabs}
\pgfplotstabletypesetfile[
    every head row/.style={
        before row=\toprule,after row=\midrule},
    every last row/.style={
        after row=\bottomrule},
    skip rows between index={2}{4},
    skip rows between index={7}{9}
]
    {pgfplotstable.example1.dat}
```

`/pgfplots/table/select equal part entry of={`⟨*part no*⟩`}{`⟨*part count*⟩`}` (style, no default)

A style which overwrites `row predicate` with a subset selection predicate. The idea is to split the current column into `{`⟨*part count*⟩`}` equally sized parts and select only `{`⟨*part no*⟩`}`.

This can be used to simulate multicolumn tables.

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A5	B5
A6	B6
A7	B7
A8	B8
A9	B9
A10	B10
A11	B11
A12	B12
A13	B13
A14	B14
A15	B15
A16	B16
A17	B17
A18	B18
A19	B19
A20	B20
A21	B21

A	B	A	B
A1	B1	A12	B12
A2	B2	A13	B13
A3	B3	A14	B14
A4	B4	A15	B15
A5	B5	A16	B16
A6	B6	A17	B17
A7	B7	A18	B18
A8	B8	A19	B19
A9	B9	A20	B20
A10	B10	A21	B21
A11	B11		

```
% requires \usepackage{booktabs}
\pgfplotstableset{
    every head row/.style={before row=\toprule,after row=\midrule},
    every last row/.style={after row=\bottomrule}}

\pgfplotstabletypesetfile[string type]{pgfplotstable.example2.dat}

\pgfplotstabletypesetfile[
    display columns/0/.style={select equal part entry of={0}{2},string type},
    display columns/1/.style={select equal part entry of={0}{2},string type},
    display columns/2/.style={select equal part entry of={1}{2},string type},
    display columns/3/.style={select equal part entry of={1}{2},string type},
    columns={A,B,A,B}
]
    {pgfplotstable.example2.dat}
```

The example above shows the original file as-is on the left side. The right side shows columns A,B,A,B – but only half of the elements are shown, selected by indices #0 or #1 of #2. The parts are equally large, up to a remainder.

If the available number of rows is not dividable by `{`⟨*part count*⟩`}`, the remaining entries are distributed equally among the first parts.

5.3 Postprocessing cells

`/pgfplots/table/postproc cell content/.code={`⟨`...`⟩`}`

Allows to *modify* assigned cell content *after* it has been assigned, possibly content-dependend. Ideas could be to draw negative numbers in red, typeset single entries in bold face or insert replacement text.

This key is evaluated *after* `assign cell content`. Its semantics is to modify an existing `@cell content` value.

There may be more than one `postproc cell content` command, if you use `.append code` or `.append style` to define them:

dof	info
4	**48€**
16	**25€**
64	**41€**
256	**8€**
1,024	**22€**
4,096	**46€**
16,384	**40€**
65,536	**48€**
$2.62 \cdot 10^{5}$	**33€**
$1.05 \cdot 10^{6}$	**2€**

```
% requires \usepackage{eurosym}
\pgfplotstabletypesetfile[
    column type=r,
    columns={dof,info},
    columns/info/.style={
        postproc cell content/.append style={
            /pgfplots/table/@cell content/.add={$\bf}{$}},
        postproc cell content/.append style={
            /pgfplots/table/@cell content/.add={}{\EUR{}}}
    }]
    {pgfplotstable.example1.dat}
```

The code above modifies `@cell content` successively. First, "`$\bf ... $`" is inserted, then "`...\EUR`". It should be noted that `pgfkeys` handles `.style` and `.code` in (quasi) the same way – both are simple code keys and can be used as such. You can combine both with `.append style` and `.append code`. Please refer to [1, section about pgfkeys] for details.

As in `assign cell contents`, the code can evaluate helper macros like `\pgfplotstablerow` to change only particular entries. Furthermore, the argument "`#1`" expands to the *unformatted* input argument which was found in the input table. This allows complete context based formatting options. Please remember that empty strings may appear due to column balancing – introduce special treatment if necessary.

There is one special case which occurs if `@cell content` itsself contains the cell separation character '`&`'. In this case, `postproc cell contents` is invoked *separately* for each part before and after the ampersand and the ampersand is inserted afterwards. This allows compatibility with special styles which create artificial columns in the output (which is allowed, see `dec sep align`). To allow separate treatment of each part, you can use the macro `\pgfplotstablepartno`. It is defined only during the evaluation of `postproc cell content` and it evaluates to the current part index (starting with 0). If there is no ampersand in your text, the value will always be 0.

This key is evaluated inside of a local TEX group, so any local macro assignments will be cleared afterwards.

The following example can be used to insert a dash, −, in a slope column:

dof	error1	slopes1
4	$2.50 \cdot 10^{-1}$	–
16	$6.25 \cdot 10^{-2}$	−1
64	$1.56 \cdot 10^{-2}$	−1
256	$3.91 \cdot 10^{-3}$	−1
1,024	$9.77 \cdot 10^{-4}$	−1
4,096	$2.44 \cdot 10^{-4}$	−1
16,384	$6.10 \cdot 10^{-5}$	−1
65,536	$1.53 \cdot 10^{-5}$	−1
$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$	−1
$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$	−1

```
\pgfplotstableset{
    create on use/slopes1/.style=
        {create col/gradient loglog={dof}{error1}}}

\pgfplotstabletypesetfile[
    columns={dof,error1,slopes1},
    columns/error1/.style={sci,sci zerofill},
    columns/slopes1/.style={
        postproc cell content/.append code={%
            \ifnum\pgfplotstablerow=0
                \pgfkeyssetvalue{/pgfplots/table/@cell content}{\ensuremath{-}}%
            \fi
        }%
    }]
    {pgfplotstable.example1.dat}
```

Since this may be useful in a more general context, it is available as `empty cells with` style.

/pgfplots/table/empty cells with={⟨*replacement*⟩} (style, no default)

Appends code to `postproc cell content` which replaces any empty cell with {⟨*replacement*⟩}.

If `dec sep align` is active, the replacement will be inserted only for the part before the decimal separator.

/pgfplots/table/set content={⟨*content*⟩} (style, no default)

A style which redefines `postproc cell contents` to always return the value {⟨*content*⟩}.

6 Customizing and getting the tabular code

/pgfplots/table/every table={⟨*file name*⟩} (style, no default)

A style which is installed at the beginning of every `\pgfplotstabletypeset` command.

The table file name is given as first argument.

/pgfplots/table/font={⟨*font name*⟩} (no default, initially `empty`)

Assigns a font used for the complete table.

/pgfplots/table/begin table={⟨*code*⟩} (no default, initially `\begin{tabular}`)

Contains {⟨*code*⟩} which is generated as table start.

/pgfplots/table/end table={⟨*code*⟩} (no default, initially `\end{tabular}`)

Contains {⟨*code*⟩} which is generated as table end.

/pgfplots/table/outfile={⟨*file name*⟩} (no default, initially `empty`)

Writes the generated tabular code into {⟨*file name*⟩}. It can then be used with `\input{`⟨*file name*⟩`}`, PgfplotsTable is no longer required since it contains a completely normal `tabular`.

dof	error1
4	0.25
16	$6.25 \cdot 10^{-2}$
64	$1.56 \cdot 10^{-2}$
256	$3.91 \cdot 10^{-3}$
1,024	$9.77 \cdot 10^{-4}$
4,096	$2.44 \cdot 10^{-4}$
16,384	$6.1 \cdot 10^{-5}$
65,536	$1.53 \cdot 10^{-5}$
$2.62 \cdot 10^{5}$	$3.81 \cdot 10^{-6}$
$1.05 \cdot 10^{6}$	$9.54 \cdot 10^{-7}$

```
\pgfplotstabletypesetfile[
    columns={dof,error1},
    outfile=pgfplotstable.example1.out.tex]
    {pgfplotstable.example1.dat}
```

and `pgfplotstable.example1.out.tex` contains

```
\begin {tabular}{cc}dof&error1\\
\pgfutilensuremath {4}&\pgfutilensuremath {0.25}\\
\pgfutilensuremath {16}&\pgfutilensuremath {6.25\cdot 10^{-2}}\\
\pgfutilensuremath {64}&\pgfutilensuremath {1.56\cdot 10^{-2}}\\
\pgfutilensuremath {256}&\pgfutilensuremath {3.91\cdot 10^{-3}}\\
\pgfutilensuremath {1{,}024}&\pgfutilensuremath {9.77\cdot 10^{-4}}\\
\pgfutilensuremath {4{,}096}&\pgfutilensuremath {2.44\cdot 10^{-4}}\\
\pgfutilensuremath {16{,}384}&\pgfutilensuremath {6.1\cdot 10^{-5}}\\
\pgfutilensuremath {65{,}536}&\pgfutilensuremath {1.53\cdot 10^{-5}}\\
\pgfutilensuremath {2.62\cdot 10^{5}}&\pgfutilensuremath {3.81\cdot 10^{-6}}\\
\pgfutilensuremath {1.05\cdot 10^{6}}&\pgfutilensuremath {9.54\cdot 10^{-7}}\\
\end {tabular}
```

The command `\pgfutilensuremath` checks whether math mode is active and switches to math mode if necessary[2].

[2]Please note that `\pgfutilensuremath` needs to be replaced by `\ensuremath` if you want to use the output file independent of PGF. That can be done by `\let\pgfutilensuremath=\ensuremath` which enables the LaTeX-command `\ensuremath`.

/pgfplots/table/debug={⟨*boolean*⟩} (no default, initially `false`)

If enabled, will write every final tabular code to your log file.

7 Defining column types for `tabular`

Besides input of text files, it is sometimes desireable to define column types for existing `tabular` environments.

\newcolumntype{⟨*letter*⟩}[⟨*number of arguments*⟩]>{⟨*before column*⟩}⟨*column type*⟩<{⟨*after column*⟩}

The command `\newcolumntype` is part of the `array` package and it defines a new column type {⟨*letter*⟩} for use in LaTeX tabular environments.

```
\usepackage{array}
```

-a+	b
-c+	d

```
\newcolumntype{d}{>{-}c<{+}}
\begin{tabular}{dl}
a & b \\
c & d \\
\end{tabular}
```

Now, the environment `pgfplotstablecoltype` can be used in {⟨*before column*⟩} and {⟨*after column*⟩} to define numerical columns:

9	2.50_{-1}
25	6.25_{-2}
81	1.56_{-2}
289	3.91_{-3}
1,089	9.77_{-4}
4,225	2.44_{-4}
16,641	6.10_{-5}
66,049	1.53_{-5}
263,169	3.81_{-6}
1,050,625	9.54_{-7}

```
% requires \usepackage{array}
\newcolumntype{L}[1]
    {>{\begin{pgfplotstablecoltype}[#1]}r<{\end{pgfplotstablecoltype}}}

\begin{tabular}{L{int detect}L{sci,sci subscript,sci zerofill}}
9       & 2.50000000e-01\\
25      & 6.25000000e-02\\
81      & 1.56250000e-02\\
289     & 3.90625000e-03\\
1089    & 9.76562500e-04\\
4225    & 2.44140625e-04\\
16641   & 6.10351562e-05\\
66049   & 1.52587891e-05\\
263169 & 3.81469727e-06\\
1050625& 9.53674316e-07\\
\end{tabular}
```

The environment `pgfplotstablecoltype` accepts an optional argument which may contain any number formatting options. It is an error if numerical columns contain non-numerical data, so it may be necessary to use `\multicolumn` for column names.

Dof	Error
9	2.50_{-1}
25	6.25_{-2}
81	1.56_{-2}
289	3.91_{-3}
1,089	9.77_{-4}
4,225	2.44_{-4}
16,641	6.10_{-5}
66,049	1.53_{-5}
263,169	3.81_{-6}
1,050,625	9.54_{-7}

```
% requires \usepackage{array}
\newcolumntype{L}[1]
    {>{\begin{pgfplotstablecoltype}[#1]}r<{\end{pgfplotstablecoltype}}}

\begin{tabular}{L{int detect}L{sci,sci subscript,sci zerofill}}
\multicolumn{1}{r}{Dof} & \multicolumn{1}{r}{Error}\\
9       & 2.50000000e-01\\
25      & 6.25000000e-02\\
81      & 1.56250000e-02\\
289     & 3.90625000e-03\\
1089    & 9.76562500e-04\\
4225    & 2.44140625e-04\\
16641   & 6.10351562e-05\\
66049   & 1.52587891e-05\\
263169 & 3.81469727e-06\\
1050625& 9.53674316e-07\\
\end{tabular}
```

8 Number formatting options

The following extract of [1] explains how to configure number formats.

`\pgfmathprintnumber{`⟨*x*⟩`}`

Generates pretty-printed output for the (real) number `{`⟨*x*⟩`}`. The input number `{`⟨*x*⟩`}` is parsed using `\pgfmathfloatparsenumber` which allows arbitrary precision.

Numbers are typeset in math mode using the current set of number printing options, see below. Optional arguments can also be provided using `\pgfmathprintnumber[`⟨*options*⟩`]{`⟨*x*⟩`}`.

`\pgfmathprintnumberto{`⟨*x*⟩`}{`⟨*\macro*⟩`}`

Returns the resulting number into `{`⟨*\macro*⟩`}` instead of typesetting it directly.

`/pgf/number format/fixed` (no value)

Configures `\pgfmathprintnumber` to round the number to a fixed number of digits after the period, discarding any trailing zeros.

4.57 0 0.1 24,415.98 123,456.12

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

See section 8.1 for how to change the appearance.

`/pgf/number format/fixed zerofill=`{⟨*boolean*⟩} (default `true`)

Enables or disables zero filling for any number drawn in fixed point format.

4.57 0.00 0.10 24,415.98 123,456.12

```
\pgfkeys{/pgf/number format/.cd,fixed,fixed zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

This key affects numbers drawn with `fixed` or `std` styles (the latter only if no scientific format is choosen).

4.57 $5 \cdot 10^{-5}$ 1.00 $1.23 \cdot 10^{5}$

```
\pgfkeys{/pgf/number format/.cd,std,fixed zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-05}\hspace{1em}
\pgfmathprintnumber{1}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

See section 8.1 for how to change the appearance.

`/pgf/number format/sci` (no value)

Configures `\pgfmathprintnumber` to display numbers in scientific format, that means sign, mantisse and exponent (basis 10). The mantisse is rounded to the desired precision.

$4.57 \cdot 10^{0}$ $5 \cdot 10^{-4}$ $1 \cdot 10^{-1}$ $2.44 \cdot 10^{4}$ $1.23 \cdot 10^{5}$

```
\pgfkeys{/pgf/number format/.cd,sci,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

See section 8.1 for how to change the exponential display style.

`/pgf/number format/sci zerofill=`{⟨*boolean*⟩} (default `true`)

Enables or disables zero filling for any number drawn in scientific format.

$4.57 \cdot 10^{0}$ $5.00 \cdot 10^{-4}$ $1.00 \cdot 10^{-1}$ $2.44 \cdot 10^{4}$ $1.23 \cdot 10^{5}$

```
\pgfkeys{/pgf/number format/.cd,sci,sci zerofill,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

As with `fixed zerofill`, this option does only affect numbers drawn in `sci` format (or `std` if the scientific format is chosen).

See section 8.1 for how to change the exponential display style.

`/pgf/number format/zerofill=`{⟨*boolean*⟩} (style, default `true`)

Sets both, `fixed zerofill` and `sci zerofill` at once.

`/pgf/number format/std` (no value)

Configures `\pgfmathprintnumber` to a standard algorithm. It chooses either `fixed` or `sci`, depending on the order of magnitude. Let $n = s \cdot m \cdot 10^{e}$ be the input number and p the current precision. If $-p/2 \leq e \leq 4$, the number is displayed using the fixed format. Otherwise, it is displayed using the scientific format.

4.57 $5 \cdot 10^{-4}$ 0.1 24,415.98 $1.23 \cdot 10^{5}$

```
\pgfkeys{/pgf/number format/.cd,std,precision=2}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

`/pgf/number format/int detect` (no value)

Configures `\pgfmathprintnumber` to detect integers automatically. If the input number is an integer, no period is displayed at all. If not, the scientific format is chosen.

15 20 $2.04 \cdot 10^{1}$ $1 \cdot 10^{-2}$ 0

```
\pgfkeys{/pgf/number format/.cd,int detect,precision=2}
\pgfmathprintnumber{15}\hspace{1em}
\pgfmathprintnumber{20}\hspace{1em}
\pgfmathprintnumber{20.4}\hspace{1em}
\pgfmathprintnumber{0.01}\hspace{1em}
\pgfmathprintnumber{0}
```

`/pgf/number format/int trunc` (no value)

Truncates every number to integers (discards any digit after the period).

4 0 0 24,415 123,456

```
\pgfkeys{/pgf/number format/.cd,int trunc}
\pgfmathprintnumber{4.568}\hspace{1em}
\pgfmathprintnumber{5e-04}\hspace{1em}
\pgfmathprintnumber{0.1}\hspace{1em}
\pgfmathprintnumber{24415.98123}\hspace{1em}
\pgfmathprintnumber{123456.12345}
```

`/pgf/number format/precision=`{⟨*number*⟩} (no default)

Sets the desired rounding precision for any display operation. For scientific format, this affects the mantisse.

8.1 Changing display styles

You can change the way how numbers are displayed. For example, if you use the '`fixed`' style, the input number is rounded to the desired precision and the current fixed point display style is used to typeset the number. The same is applied to any other format: first, rounding routines are used to get the correct digits, afterwards a display style generates proper TEX-code.

`/pgf/number format/set decimal separator={`⟨*text*⟩`}` (no default)

Assigns `{`⟨*text*⟩`}` as decimal separator for any fixed point numbers (including the mantisse in sci format).

`/pgf/number format/dec sep={`⟨*text*⟩`}` (no default)

Just another name for `set decimal separator`.

`/pgf/number format/set thousands separator={`⟨*text*⟩`}` (no default)

Assigns `{`⟨*text*⟩`}` as thousands separator for any fixed point numbers (including the mantisse in sci format).

1234.56

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,
    precision=2,
    set thousands separator={}}
\pgfmathprintnumber{1234.56}
```

1234567890.00

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,
    precision=2,
    set thousands separator={}}
\pgfmathprintnumber{1234567890}
```

1.234.567.890.00

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,
    precision=2,
    set thousands separator={.}}
\pgfmathprintnumber{1234567890}
```

1, 234, 567, 890.00

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,
    precision=2,
    set thousands separator={,}}
\pgfmathprintnumber{1234567890}
```

1,234,567,890.00

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,
    precision=2,
    set thousands separator={{{{,}}}}}
\pgfmathprintnumber{1234567890}
```

The last example employs commas and disables the default comma-spacing.

`/pgf/number format/1000 sep={`⟨*text*⟩`}` (no default)

Just another name for `set thousands separator`.

`/pgf/number format/use period` (no value)

A predefined style which installs periods '`.`' as decimal separators and commas '`,`' as thousands separators. This style is the default.

12.35

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2,use period}
\pgfmathprintnumber{12.3456}
```

1,234.56

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2,use period}
\pgfmathprintnumber{1234.56}
```

`/pgf/number format/use comma` (no value)

A predefined style which installs commas ‘`,`’ as decimal separators and periods ‘`.`’ as thousands separators.

12,35

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2,use comma}
\pgfmathprintnumber{12.3456}
```

1.234,56

```
\pgfkeys{/pgf/number format/.cd,fixed,precision=2,use comma}
\pgfmathprintnumber{1234.56}
```

`/pgf/number format/skip 0.=`{⟨*boolean*⟩} (no default, initially `false`)

Configures whether numbers like 0.1 shall be typeset as .1 or not.

.56

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,precision=2,
    skip 0.}
\pgfmathprintnumber{0.56}
```

0.56

```
\pgfkeys{/pgf/number format/.cd,
    fixed,
    fixed zerofill,precision=2,
    skip 0.=false}
\pgfmathprintnumber{0.56}
```

`/pgf/number format/showpos=`{⟨*boolean*⟩} (no default, initially `false`)

Enables or disables display of plus signs for non-negative numbers.

+12.35

```
\pgfkeys{/pgf/number format/showpos}
\pgfmathprintnumber{12.345}
```

12.35

```
\pgfkeys{/pgf/number format/showpos=false}
\pgfmathprintnumber{12.345}
```

$+1.23 \cdot 10^{1}$

```
\pgfkeys{/pgf/number format/.cd,showpos,sci}
\pgfmathprintnumber{12.345}
```

`/pgf/number format/print sign=`{⟨*boolean*⟩} (style, no default)

A style which is simply an alias for `showpos=`{⟨*boolean*⟩}.

`/pgf/number format/sci 10e` (no value)

Uses $m \cdot 10^{e}$ for any number displayed in scientific format.

$1.23 \cdot 10^{1}$

```
\pgfkeys{/pgf/number format/.cd,sci,sci 10e}
\pgfmathprintnumber{12.345}
```

`/pgf/number format/sci 10^e` (no value)

The same as ‘`sci 10e`’.

`/pgf/number format/sci e` (no value)

Uses the ‘1*e*+0’ format which is generated by common scientific tools for any number displayed in scientific format.

1.23*e*+1

```
\pgfkeys{/pgf/number format/.cd,sci,sci e}
\pgfmathprintnumber{12.345}
```

/pgf/number format/sci E (no value)

The same with an uppercase 'E'.

$1.23E+1$

```
\pgfkeys{/pgf/number format/.cd,sci,sci E}
\pgfmathprintnumber{12.345}
```

/pgf/number format/sci subscript (no value)

Typesets the exponent as subscript for any number displayed in scientific format. This style requires very few space.

1.23_1

```
\pgfkeys{/pgf/number format/.cd,sci,sci subscript}
\pgfmathprintnumber{12.345}
```

/pgf/number format/@dec sep mark={⟨*text*⟩} (no default)

Will be placed right before the place where a decimal separator belongs to. However, {⟨*text*⟩} will be inserted even if there is no decimal separator. It is intented as place-holder for auxiliary routines to find alignment positions.

This key should never be used to change the decimal separator! Use `dec sep` instead.

/pgf/number format/@sci exponent mark={⟨*text*⟩} (no default)

Will be placed right before exponents in scientific notation. It is intented as place-holder for auxiliary routines to find alignment positions.

This key should never be used to change the exponent!

/pgf/number format/assume math mode={⟨*boolean*⟩} (default `true`)

Set this to `true` if you don't want any checks for math mode.

The initial setting installs a `\pgfutilensuremath` around each final number to change to math mode if necessary. Use `assume math mode=true` if you know that math mode is active and you don't want `\pgfutilensuremath`.

9 Changing tables

After tables have been loaded from disk, it is possible to change their contents. The methods are limited up to now and they reflect mostly what I needed in my applications. Nevertheless, the approaches are quite general and can be customized.

\pgfplotstablecreatecol[⟨*options*⟩]{⟨*new col name*⟩}{⟨*table*⟩}

Creates a new column named {⟨*new col name*⟩} and appends it to table {⟨*table*⟩}.

This command is technical, but it offers a flexible framework to generate new columns. It has been designed to create new columns using the already existing values – for example using logical or numerical methods to combine existing values. It provides fast access to a row's value and the next row's value.

The following documentation is for all who want to *write* specialised columns. It is not particularly difficult; it is just technical and it requires some knowledge of `pgfkeys`. If you don't like it, you can resort to some predefined column generation styles - and enable those styles in {⟨*options*⟩}.

The column entries will be created using the command key `create col/assign`. It will be invoked for every row of the table. It is supposed to assign contents to `create col/next content`. During the evaluation, the macro `\thisrow`{⟨*col name*⟩} expands to the current row's value of the column identified by {⟨*col name*⟩}. Furthermore, `\nextrow`{⟨*col name*⟩} expands to the *next* row's value of the designated column and `\prevrow`{⟨*col name*⟩} expands to the value of the *previous* row.

Two special `assign` routines are available for the first and last row: The contents for the *last* row is computed with `create col/assign last`. Its semantics is the same. The contents for the *first* row is computed with `create col/assign first` to simplify special cases here. These first and last commands are optional, their default is to invoke the normal `assign` routine.

The evaluation of the `assign` keys is done in local TEX groups.

The following macros are useful during cell assignments:

1. `\prevrow{`⟨*col name*⟩`}` / `\getprevrow{`⟨*col name*⟩`}{`⟨*\macro*⟩`}`
   These two routines return the value stored in the *previous* row of the designated column `{`⟨*col name*⟩`}`. The `get` routine stores it into ⟨*\macro*⟩.
2. `\thisrow{`⟨*col name*⟩`}` / `\getthisrow{`⟨*col name*⟩`}{`⟨*\macro*⟩`}`
   These two routines return the *current* row's value stored in the designated column. The `get` routine stores it into ⟨*\macro*⟩.
3. `\nextrow{`⟨*col name*⟩`}` / `\getnextrow{`⟨*col name*⟩`}{`⟨*\macro*⟩`}`
   These two routines return the *next* row's value.
4. `\pgfplotstablerow` and `\pgfplotstablerows` which contain the current row's index and the total number of rows, respectively. See page 8 for details.
5. `\pgfmathaccuma` and `\pgfmathaccumb` can be used to transport intermediate results. Both maintain their value from one call to the next. All other local variables will be deleted after leaving the assignment routines.

The `{`⟨*col name*⟩`}` is expected to be a *physical* column name, no alias or column index is allowed (unless column indices and column names are the same).

The following example takes our well-known input table and creates a copy of the `level` column. Furthermore, it produces a lot of output to show the available macros. Finally, it uses `\pgfkeyslet` to assign the contents of the resulting `\entry` to `next content`.

level	new
1	thisrow=1; nextrow=2. (#0/10)
2	thisrow=2; nextrow=3. (#1/10)
3	thisrow=3; nextrow=4. (#2/10)
4	thisrow=4; nextrow=5. (#3/10)
5	thisrow=5; nextrow=6. (#4/10)
6	thisrow=6; nextrow=7. (#5/10)
7	thisrow=7; nextrow=8. (#6/10)
8	thisrow=8; nextrow=9. (#7/10)
9	thisrow=9; nextrow=10. (#8/10)
10	thisrow=10; nextrow=. (#9/10)

```
\pgfplotstableread{pgfplotstable.example1.dat}\table
\pgfplotstablecreatecol[
    create col/assign/.code={%
        \getthisrow{level}\entry
        \getnextrow{level}\nextentry
        \edef\entry{thisrow=\entry; nextrow=\nextentry.
            (\#\pgfplotstablerow/\pgfplotstablerows)}%
        \pgfkeyslet{/pgfplots/table/create col/next content}\entry
    }]
    {new}\table

\pgfplotstabletypeset[
    column type=l,
    columns={level,new},
    columns/new/.style={string type}
]\table
```

There is one more speciality: you can use `columns={`⟨*column list*⟩`}` to reduce the runtime complexity of this command. This works only if the `columns` key is provided directly into `{`⟨*options*⟩`}`. In this case `\thisrow` and its variants are only defined for those columns listed in the `columns` value.

Limitations. Currently, you can only access three values of one column at a time: the current row, the previous row and the next row. Access to arbitrary indices is not (yet) supported. Furthermore, this command has been designed to modify an existing table. You can't create a table from scratch with this command.

The default implementation of `assign` is to produce empty strings. The default implementation of `assign last` is to invoke `assign`, so in case you never really use the next row's value, you won't need to touch `assign last`. The same holds for `assign first`.

/pgfplots/table/create on use/⟨*col name*⟩/.style={⟨*create options*⟩}

Allows "lazy creation" of the column ⟨*col name*⟩. Whenever the column ⟨*col name*⟩ is queried by name, for example in an `\pgfplotstabletypeset` command, and such a column does not exist already, it is created on-the-fly.

error1	quot1
$2.50 \cdot 10^{-1}$	
$6.25 \cdot 10^{-2}$	4
$1.56 \cdot 10^{-2}$	4
$3.91 \cdot 10^{-3}$	4
$9.77 \cdot 10^{-4}$	4
$2.44 \cdot 10^{-4}$	4
$6.10 \cdot 10^{-5}$	4
$1.53 \cdot 10^{-5}$	4
$3.81 \cdot 10^{-6}$	4
$9.54 \cdot 10^{-7}$	4

```
% requires \usepackage{array}
\pgfplotstableset{
    create on use/quot1/.style=
        {create col/quotient={error1}}}

\pgfplotstabletypesetfile[
    columns={error1,quot1},
    columns/error1/.style={sci,sci zerofill},
    columns/quot1/.style={dec sep align}]
{pgfplotstable.example1.dat}
```

The example above queries `quot1` which does not yet exist in the input file. Therefor, it is checked whether a `create on use` style for `quot1` exists. This is the case, so it is used to create the missing column.

A `create on use` specification is translated into

`\pgfplotstablecreatecol[`⟨*create options*⟩`]{`⟨*col name*⟩`}{`⟨*the table*⟩`}`.

This feature allows some lazyness, because you can omit the lengthy table modifications. However, lazyness may cost something: in the example above, the generated column will be *lost* after returning from `\pgfplotstabletypesetfile`.

The `create on use` has higher priority than `alias`.

In case ⟨*col name*⟩ contains characters which are required for key settings, you need to use braces around it: "`create on use/{name=wi/th,special}/.style={...}`".

/pgfplots/table/create col/expr={⟨*math expression*⟩} (style, no default)

A style for use in `\pgfplotstablecreatecol` which uses `{`⟨*math expression*⟩`}` to assign contents for the new column.

level	2·level
1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18
10	20

```
\pgfplotstableread{pgfplotstable.example1.dat}\table
\pgfplotstablecreatecol[
    create col/expr={\thisrow{level}*2}]
    {new}\table

\pgfplotstabletypeset[
    columns={level,new},
    columns/new/.style={column name=$2\cdot $level}
]\table
```

The macros `\thisrow{`⟨*col name*⟩`}` and `\nextrow{`⟨*col name*⟩`}` can be used to use values of the existing table.

Please see `\pgfplotstablecreatecol` for more information.

Attention: Currently, `expr` uses the math parser of PGF. Although very powerful, it has never been intended for scientific applications. Its 5.5 fixed point arithmetic has its limitations.

There is limited support for float point arithmetics, but that is far from complete and every elementary operation needs to be programmed by hand. Some of the following numerical styles employ these routines, they have been included into the PGF math library (but not into the parser).

Furthermore, you can create new columns using the `fp` package which allows precise fixed point arithmetics.

/pgfplots/table/create col/quotient={⟨*column name*⟩} (style, no default)

A style for use in `\pgfplotstablecreatecol` which computes the quotient $c_i := m_{i-1}/m_i$ for every entry $i = 1, \dots, (n-1)$ in the column identified with {⟨*column name*⟩}. The first value c_0 is kept empty.

error1	error2	quot1	quot2
$2.50 \cdot 10^{-1}$	$7.58 \cdot 10^{-1}$		
$6.25 \cdot 10^{-2}$	$5.00 \cdot 10^{-1}$	4	1.52
$1.56 \cdot 10^{-2}$	$2.87 \cdot 10^{-1}$	4	1.74
$3.91 \cdot 10^{-3}$	$1.44 \cdot 10^{-1}$	4	2
$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	4	3.25
$2.44 \cdot 10^{-4}$	$1.70 \cdot 10^{-2}$	4	2.6
$6.10 \cdot 10^{-5}$	$8.20 \cdot 10^{-3}$	4	2.07
$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	4	2.1
$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	4	2
$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	4	2

```
% requires \usepackage{array}
\pgfplotstableread{pgfplotstable.example1.dat}\table
\pgfplotstablecreatecol[create col/quotient=error1]
    {quot1}\table
\pgfplotstablecreatecol[create col/quotient=error2]
    {quot2}\table

\pgfplotstabletypeset[
    columns={error1,error2,quot1,quot2},
    columns/error1/.style={sci,sci zerofill},
    columns/error2/.style={sci,sci zerofill},
    columns/quot1/.style={dec sep align},
    columns/quot2/.style={dec sep align}]
\table
```

This style employs methods of the floating point unit, that means it works with a relative precision of about 10^{-7} (7 significant digits in the mantisse).

/pgfplots/table/create col/iquotient={⟨*column name*⟩} (style, no default)

Like `create col/quotient`, but the quotient is inverse.

/pgfplots/table/create col/dyadic refinement rate={⟨*column name*⟩} (style, no default)

A style for use in `\pgfplotstablecreatecol` which computes the convergence rate α of the data in column {⟨*column name*⟩}. The contents of {⟨*column name*⟩} is assumed to be something like $e_i(h_i) = O(h_i^\alpha)$. Assuming a dyadic refinement relation from one row to the next, $h_i = h_{i-1}/2$, we have $h_{i-1}^\alpha/(h_{i-1}/2)^\alpha = 2^\alpha$, so we get α using

$$c_i := \log_2 \left(\frac{e_{i-1}}{e_i} \right).$$

The first value c_0 is kept empty.

error1	error2	rate1	rate2
$2.50 \cdot 10^{-1}$	$7.58 \cdot 10^{-1}$		
$6.25 \cdot 10^{-2}$	$5.00 \cdot 10^{-1}$	2	0.6
$1.56 \cdot 10^{-2}$	$2.87 \cdot 10^{-1}$	2	0.8
$3.91 \cdot 10^{-3}$	$1.44 \cdot 10^{-1}$	2	1
$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	2	1.7
$2.44 \cdot 10^{-4}$	$1.70 \cdot 10^{-2}$	2	1.38
$6.10 \cdot 10^{-5}$	$8.20 \cdot 10^{-3}$	2	1.05
$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	2	1.07
$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	2	1
$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	2	1

```
% requires \usepackage{array}
\pgfplotstabletypesetfile[
    create on use/rate1/.style={create col/dyadic refinement rate={error1}},
    create on use/rate2/.style={create col/dyadic refinement rate={error2}},
    columns={error1,error2,rate1,rate2},
    columns/error1/.style={sci,sci zerofill},
    columns/error2/.style={sci,sci zerofill},
    columns/rate1/.style={dec sep align},
    columns/rate2/.style={dec sep align}]
    {pgfplotstable.example1.dat}
```

This style employs methods of the floating point unit, that means it works with a relative precision of about 10^{-6} (6 significant digits in the mantisse).

`/pgfplots/table/create col/idyadic refinement rate={`⟨*column name*⟩`}` (style, no default)

As `create col/dyadic refinement rate`, but the quotient is inverse.

`/pgfplots/table/create col/gradient={`⟨*col x*⟩`}{`⟨*col y*⟩`}` (no default)
`/pgfplots/table/create col/gradient loglog={`⟨*col x*⟩`}{`⟨*col y*⟩`}` (no default)
`/pgfplots/table/create col/gradient semilogx={`⟨*col x*⟩`}{`⟨*col y*⟩`}` (no default)
`/pgfplots/table/create col/gradient semilogy={`⟨*col x*⟩`}{`⟨*col y*⟩`}` (no default)

A style for `\pgfplotstablecreatecol` which computes piecewise gradients $(y_{i+1} - y_i)/(x_{i+1} - x_i)$ for each row. The y values are taken out of column `{`⟨*col y*⟩`}` and the x values are taken from `{`⟨*col y*⟩`}`.

The logarithmic variants apply the natural logarithm, $\log(\cdot)$, to its argument before starting to compute differences. More precisely, the `loglog` variant applies the logarithm to both, x and y, the `semilogx` variant applies the logarithm only to x and the `semilogy` variant applies the logarithm only to y.

dof	error1	error2	slopes1	slopes2
4	$2.50 \cdot 10^{-1}$	$7.58 \cdot 10^{-1}$		
16	$6.25 \cdot 10^{-2}$	$5.00 \cdot 10^{-1}$	-1	-0.3
64	$1.56 \cdot 10^{-2}$	$2.87 \cdot 10^{-1}$	-1	-0.4
256	$3.91 \cdot 10^{-3}$	$1.44 \cdot 10^{-1}$	-1	-0.5
1,024	$9.77 \cdot 10^{-4}$	$4.42 \cdot 10^{-2}$	-1	-0.85
4,096	$2.44 \cdot 10^{-4}$	$1.70 \cdot 10^{-2}$	-1	-0.69
16,384	$6.10 \cdot 10^{-5}$	$8.20 \cdot 10^{-3}$	-1	-0.53
65,536	$1.53 \cdot 10^{-5}$	$3.91 \cdot 10^{-3}$	-1	-0.53
262,144	$3.81 \cdot 10^{-6}$	$1.95 \cdot 10^{-3}$	-1	-0.5
1,048,576	$9.54 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	-1	-0.5

```
% requires \usepackage{array}
\pgfplotstableread{pgfplotstable.example1.dat}\table
\pgfplotstablecreatecol[create col/gradient loglog={dof}{error1}]
    {slopes1}\table
\pgfplotstablecreatecol[create col/gradient loglog={dof}{error2}]
    {slopes2}\table

\pgfplotstabletypeset[
    columns={dof,error1,error2,slopes1,slopes2},
    columns/dof/.style={int detect},
    columns/error1/.style={sci,sci zerofill},
    columns/error2/.style={sci,sci zerofill},
    columns/slopes1/.style={dec sep align},
    columns/slopes2/.style={dec sep align}]
\table
```

level	error1	slopes1
1	2.50_{-1}	
2	6.25_{-2}	-1.39
3	1.56_{-2}	-1.39
4	3.91_{-3}	-1.39
5	9.77_{-4}	-1.39
6	2.44_{-4}	-1.39
7	6.10_{-5}	-1.39
8	1.53_{-5}	-1.39
9	3.81_{-6}	-1.39
10	9.54_{-7}	-1.39

```
% requires \usepackage{array}
\pgfplotstableread
    {pgfplotstable.example1.dat}\table
\pgfplotstablecreatecol
    [create col/gradient semilogy={level}{error1}]
    {slopes1}\table

\pgfplotstabletypeset[
    columns={level,error1,slopes1},
    columns/level/.style={int detect},
    columns/error1/.style=
        {sci,sci zerofill,sci subscript},
    columns/slopes1/.style={dec sep align}]
\table
```

This style employs methods of the floating point unit, that means it works with a relative precision of about 10^{-6} (6 significant digits in the mantisse).

10 Plain TEX and ConTEXt support

The table code generator is initialised to produce LATEX `tabular` environments. However, it only relies on '`&`' being the column separator and '`\\`' the row terminator. The `column type` feature is more or less specific to `tabular`, but you can disable it completely. Replace `begin table` and `end table` with appropriate TEX- or ConTEXt commands to change it. If you have useful default styles (or bug reports), let me know.

Index

References

[1] T. Tantau. Ti*k*Z and PGF manual. `http://sourceforge.net/projects/pgf`. *v.* ≥ 2.00.