`numerica-tables`

version 3.2.0

Andrew Parsloe
(ajparsloe@gmail.com)

November 20, 2024

Abstract

The `numerica-tables` package enables the creation of multi-row, multi-column tables of values of mathematical functions. Key–value assignments allow presentation in a wide variety of table styles, both vertically by column or horizontally by row, within the 'formal table' framework of the `booktabs` package. `numerica-tables` requires the prior loading of the `numerica` package.

- This document applies to version 3.2.0 of `numerica-tables` and requires the `booktabs` package.

- Version 3 of `numerica` needs to be loaded before `numerica-tables`; (`numerica` requires `amsmath` and `mathtools`).

- I refer many times in this document to tables included in *Handbook of Mathematical Functions*, edited by Milton Abramowitz and Irene A. Stegun, Dover, 1965, and referenced as *HMF*.

- Version 3.2.0 of `numerica-tables`
  - adds a setting `transpose` to convert a table with function values in columns into a table with function values in rows;
  - adds the package options `rules` and `norules` to specify the default pattern of horizontal rules;
  - adds new settings and enhances existing settings to adjust the appearance, size and positioning of items in both the row variable column and the header row;
  - enhances and adjusts elements of fraction form output (e.g. with a 'semi-verbatim' setting);
  - fixes a bug when tabulating factorials;
  - updates documentation.

- Version 3.1.0 of `numerica-tables`
  - adds an index to the documentation;
  - adds the ability to round table entries to different values.

- Version 3.0.0 of `numerica-tables`
  - adds the ability to use as row variable values numbers or expressions from a comma list, a macro, a file, or a step function, to be displayed either as values or verbatim;
  - adds the ability to suppress the header row;
  - is compatible with the additional features of `numerica` version 3.0.0, including the decimal comma and fraction-form output.

Contents

Chapter 1

Introduction

Entering

```
\usepackage[<options>]{numerica}
\usepackage[<options>]{numerica-tables}
```

in the preamble of a document gives access to a command for creating tables of function values in a wide variety of styles. Contrary to previous practice, from version 3.0.0 of `numerica-tables`, the `numerica` package is not loaded automatically but must be loaded explicitly (as above), with options if desired, *before* `numerica-tables`. It is *essential* that the version of `numerica` loaded is version 3.

All tables are 'formal tables' in the sense of the `booktabs` package, which is loaded automatically. Such tables have no vertical rules and few horizontal rules. Quoting from the documentation for that package:

1. Never, ever use vertical rules.
2. Never use double rules.

1.1 Package options

New with version 3.2 of `numerica-tables` are the package options `rules` and `norules`, and `Q?*`, which is more complicated and discussed later (§2.5.6.1). The `rules` option determines which (if any) horizontal rules are drawn in a table if none are explicitly set for that table. Because most tables in this document are set amongst text, they are delimited by top and bottom rules, and one beneath the header row. For consistency with earlier versions, 'out of the box' the `rules` package option is initialized to `rules=ThB`.

For the user this may not be what is wanted. You may prefer by default to insert a rule only beneath the header row of your tables in which case the package option would be `rules=h`. For other possible values for the package option see the identical values of the `rules` *setting* discussed in §2.4.5. If, as a default, you want no rules at all in your tables, you can either give `rules` an empty value (`rules=  `) or use the `norules` package option:

```
\usepackage[norules]{numerica-tables}
```

1.2 Table structure

I take as my source of models of mathematical tables those presented in *Handbook of Mathematical Functions*, edited by Milton Abramowitz and Irene A. Stegun, Dover, 1965, not because the typesetting is elegant (it often is not) but because *HMF* displays a wide variety of table styles. The editors of that volume were faced with a host of different problems requiring a host of different solutions. The `\nmcTabulate` command of `numerica-tables` aims to reproduce most of those different solutions, within `booktabs` elegance.

Creating a table presumes we have a function or functions we wish to tabulate. The values a function takes will generally depend on a primary parameter and, possibly, a number of secondary parameters (which is where much of the complexity comes from). Books of mathematical tables are structured (nearly always) in *columns* (but see below), and we read (nearly always) *down* a column as the primary parameter is incremented, generally in regular steps. We need to decide on the range of values the primary parameter will take and how fine-grained the tabulation will be – what the step size of its increments will be. Assigning different values to a second parameter generates a second, third, ... column of function values. Sometimes rather than a second parameter, a second, third, ... function of the first parameter is tabulated in the successive columns – like adjacent columns of different trigonometric functions.

In this document the first parameter is called the *row variable* – its value determines which row we are in; the second parameter, if present, is called the *column variable* – its value determines which column we are in. A table generally (but not always) presents the values of the row variable in the first column, the *row variable column*, sometimes in distinctive type (e.g. bolded). The values of the column variable are presented in a *header row* above the table body of function values. Above the header row there may be a *title row* and perhaps a *subtitle row* where other explanatory material can be displayed. Sometimes there is a *footer row* beneath the table body. Vertical rules are absent, horizontal rules used sparingly – for example, at the top and bottom of the table, or under the header row, but not in the body of the table.

1.2.1 Transposed tables

From version 3.2 of `numerica-tables` it is also possible to present tables 'horizontally', in which function values are read in rows (generally) left-to-right across the page. `numerica-tables` handles this situation by constructing the table in the usual column-based 'vertical' manner and then treating the columns of calculated function values as rows; the row variable column and the header row swap roles. The placement of the new row variable column and header row and the ‘decoration’ of the values contained therein (font, alignment, etc.) is all done *after* transposition.

The process is effected through the setting `transpose` and is discussed at §2.4.8, but examples are presented throughout the document.

1.2.2 `tabular` environment

The tables created by `numerica-tables` use the `tabular` environment of LaTeX. This means that the vertical separation between rows can be altered by means of the parameter `\arraystretch`. Writing, for instance,

```
\renewcommand\arraystretch{1.2}
```

increases the separation between rows by a scale factor of 1.2. An example of use occurs in §2.1.3.7.

The separation between columns is determined by the parameter `\tabcolsep` which is half the width of the space between columns and defaults to 6 pt. To change `\tabcolsep` you can use either of the commands `\addtolength` or `\setlength`. Since the default is 6pt, the commands

```
\addtolength\tabcolsep{3pt}
\setlength\tabcolsep{9pt}
```

have the same effect and increase the half-width between columns to 9 pt; see the final example in §2.2.2.7.

After changing either of these parameters you will need to reset it back to the default if you do not want all subsequent tables to be stretched vertically or horizontally as the case may be.

1.3 Shared syntax

The `\nmcTabulate` command, short-name form `\tabulate`, shares the syntax of `\nmcEvaluate` (see `numerica.pdf`). When all options are used the command looks like

```
\nmcTabulate*[settings]{expr.}[vv-list][num. format]
```

1. `*` optional switch; if present ensures a single number output with no formatting, or an appropriate error message if the single number cannot be produced; see §2.5.6.1;
2. `[settings]` *mandatory* comma-separated list of *key=value* settings; this option is at the heart of creating a table of function values, and is discussed in the next chapter;
3. `{expr.}` mandatory argument specifying the mathematical expression or expressions in LaTeX form to be tabulated;

4. `[vv-list]` *mandatory* comma-separated list (or semicolon-separated list if the `comma` package option is used with `numerica`) of *variable=value* items, in particular containing the initial value of the row variable (essential) and column variable (if one is used);

5. `[num. format]` optional format specification for presentation of the numerical results (rounding, padding with zeros, scientific notation, fraction-form output); see §2.5.1.

Unlike `\nmcEvaluate` (the main command in `numerica`), for `\nmcTabulate` the two apparently optional arguments straddling the main argument (`settings` and `vv-list`) are *essential*. Although both are delimited by square brackets, that is in order to draw on the `numerica` code. Each argument contains items *necessary* for the construction of any table of function values.

Should `numerica` be loaded with the `comma` package option, numbers in tables will be displayed with a decimal comma. In this case, to avoid confusion, items in the vv-list *must* be separated by a semicolon. Similarly, *n*-ary functions – `\max`, `\min` and `\gcd` – must use semicolons as their argument separator. From version 3.2 of `numerica-tables` row variable data in the form of lists or files for the `rdata` and `rfile` settings do not need to be *comma* separated. If the `comma` package option is used, the semicolon is assumed (see §2.1.2.2) or some other item separator can be set with the (new in version 3.2) `rdelim` setting.

Although math environments are significant for `\nmcEvaluate`, they should be avoided with `\nmcTabulate`. Placing a `\tabulate` command within a math environment, or vice versa, is likely to cause a LaTeX error. From version 3.2 of `numerica-tables`, it is possible to specify the math style (`\displaystyle`, `\textstyle` or `\scriptstyle`) of entries in the row variable column and in the header row. Function values (numbers) in the table cells are presented between inline math delimiters (`$ $`).

Chapter 2

`\nmcTabulate` settings

Just as `\nmcTabulate` shares the syntax of `\nmcEvaluate` (of `numerica`), it also shares the *settings* of the latter command – although not all will be relevant. See the associated document `numerica.pdf` for a list of those settings and associated discussion. I will point out instances of their use in the following examples.

In addition to the shared settings, there are many specific to `\nmcTabulate`, which is what this document is about. They are discussed in groups in later sections, some in more than one place. For the main discussion of row variable settings, see immediately below; for column variable settings see §2.2; for multi-function tables see §2.3; for whole-of-table formatting see §2.4; for formatting the function values in table cells see §2.5.

2.1 Row variable settings

Okay, assume we have chosen the function we want tabulated. Now we need to choose the row variable, how fine-grained we want the tabulation to be and between which values.

2.1.1 Row variable specification: uniform case

The *row* variable is set in the settings option of the `\tabulate` command with the key `rvar` – say `rvar=x`. What value to start tabulating from is specified in the vv-list – `x=0` perhaps – and so does not need a specific key, but what value to tabulate to, `rstop`, and the step size, how fine-grained the tabulation is to be, `rstep`, do need to be specified in the settings option. In the uniform case (which makes up the overwhelming majority of cases in *HMF*) the step size is constant. It does not change as the value of the row variable changes. (The non-uniform case, available from version 3.0.0 of `numerica-tables`, is discussed in §2.1.2 below. Quite different keys are required.)

The two tables in the example below tabulate $\sin x$ and $\cos x$ between 0 and 1 in increments of 0.2. By placing the start value of the tabulation variable in

Table 2.1: Row variable specification

key	type	meaning	comment
`rvar`	token(s)	row variable	
`rstep`	real num.	step size	
`rstop`	real num.	stop value	excludes `rows`
`rows`	int	number of rows	excludes `rstop`
`rspec`	comma list	`{rvar, rstep, rows}`	short form spec.
`rround`	int	rounding	default: `1`

the vv-list, the possibility is opened for other parameters in more complicated functions to depend on the row variable. Although it will often be the first entry in the vv-list, it does not need to be.

The initial value of the row variable may depend on other quantities which must necessarily precede it – lie to the right of it – in the vv-list. The start value may be a LaTeX expression. Both `rstep` and `rstop` may also be LaTeX expressions. They are evaluated *after* the vv-list is evaluated and so may depend on the values of variables in the vv-list, including the *initial* value of the row variable. As of v.3.2 of `numerica-tables` the `vv@=1` (or `vvmode=1`) setting from `numerica` does not affect `rstep` and `rstop`. For instance if `rvar=x`, setting `rstep=1/x` when `vv@=1` gives a *constant* step size equal to the reciprocal of the initial value of the row variable. The following examples illustrate the use of `rvar`, `rstep` and `rstop` and the presence of the initial value of the row variable in the vv-list.

```
\tabulate[rvar=x,rstep=0.2,rstop=1]
  { \sin x }[x=0]\qquad
\tabulate[rvar=x,rstep=0.2,rstop=1]
  { \sin x }[x=0][*]
```

$\Longrightarrow$

x	$\sin x$
0	0
0.2	0.198669
0.4	0.389418
0.6	0.564642
0.8	0.717356
1	0.841471

x	$\sin x$
0.0	0.000000
0.2	0.198669
0.4	0.389418
0.6	0.564642
0.8	0.717356
1.0	0.841471

The difference in appearance of the tables results from padding with zeros in the second. The asterisk in the trailing optional argument has the same effect in `\nmcTabulate` as in `\nmcEvaluate`. As you can see, padding applies not only to the values of the function but also to the values of the row variable, although that has been padded to only 1 decimal place (the default) rather than the 6 of the function values. (How many digits to round to, and therefore pad the row variable values to, is discussed in §2.1.1.1.) Padding makes an obvious improvement to the appearance, both in the function-value column and the row variable column.

You may feel a single column of function values makes poor use of space on the page. To turn the second of these tables into a 'horizontal' table, add the key `transpose` to the settings option. I have also reduced the number of decimal places from the default 6 to 4 to avoid any issues with page width.

```
\tabulate[rvar=x,rstep=0.2,rstop=1,transpose]
  { \sin x }[x=0][4*]
```

⟹

x	0.0	0.2	0.4	0.6	0.8	1.0
$\sin x$	0.0000	0.1987	0.3894	0.5646	0.7174	0.8415

To my eye, for such a 'shallow' (vertically compressed) table, the horizontal rules here are intrusive. They can be turned off; see §2.4.5 or, anticipating the later discussion, add `norules` to the settings option (prior to version 3.2 of `numerica-tables` it was necessary to write `rules=` , an empty setting, to achieve this), and just for variety use the cosine:

```
\tabulate[rvar=x,rstep=0.2,rstop=1,transpose,norules]
  { \cos x }[x=0][4*]
```

⟹

x	0.0	0.2	0.4	0.6	0.8	1.0
$\cos x$	1.0000	0.9801	0.9211	0.8253	0.6967	0.5403

The remaining quibble is the placement of the column headers. Can they be centred or otherwise adjusted? Yes; see §2.1.3 for ways of adjusting the row variable column and header, and §2.2.2 for ways of adjusting the function-value columns and headers.

It may be more convenient at times to specify the number of rows, `rows`, to tabulate rather than a stop value. Only one of `rows` and `rstop` should be given, but if both (inadvertently) are present, it is `rows` that prevails. The first of the following three tables shows an example where `rows` is specified. The second and third tables use an abbreviated form of the row variable specification, `rspec`. This is a three-element comma list, `{rvar,rstep,rows}` (with an optional fourth element, `rround`; see §2.1.1.1). The second table gives a straightforward example. In the third table a simple LaTeX expression has been inserted for `rows` in the `rspec` comma list. Like `rstep` and `rstop`, `rows` can be a LaTeX expression but unlike `rstep` and `rstop` it is evaluated *before* the vv-list and therefore cannot depend on quantites specified there like the initial row variable value. On evaluation, the LaTeX expression is rounded to an integer.

```
\tabulate[rvar=x,rstep=0.2,rows=6]
  { \sin x/\cos x }[x=0][*] \qquad
\tabulate[rspec={x,0.2,6}]
  { \tan x }[x=0][*] \qquad
```

```
\tabulate[rspec={x,0.2,1.2/0.2}]
  { \sqrt{\sec^2 x - 1} }[x=0.2][*]
```

⟹

x	$\sin x/\cos x$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408
1.2	2.572152

x	$\tan x$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408
1.2	2.572152

x	$\sqrt{\sec^2 x - 1}$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408
1.2	2.572152

2.1.1.1 Rounding: `rround`

After studying some of the previous tables, we might decide to adjust the step size, say from 0.2 to 0.25. But changing `rstep` to the new value gives a disconcerting result, as you can see from the first of the tables below. `numerica-tables` uses a default rounding value of 1 for the row variable and has rounded 0.25 down to 0.2, then $0.2 + 0.25 = 0.45$ down to 0.4, then $0.4 + 0.25 = 0.65$ down to 0.6, then $0.6+0.25 = 0.85$ down to 0.8, at which point it stops since $0.8+0.25 > 1$ which is the stopping value. The rounded-down values are the values used – always it is the *displayed* rounded value that is used for calculating function values. The second table corrects matters by adjusting the row variable rounding with `rround=2`. From version 3.2 of `numerica-tables` the `rround` value can also be included as an optional fourth element to `rspec`. The third table illustrates this abbreviated form of specification:

```
\tabulate[rvar=x,rstep=0.25,rstop=1]
  { \exp x }[x=0][*]  \qquad
\tabulate[rvar=x,rstep=0.25,rstop=1,rround=2]
  { \exp x }[x=0][*] \qquad
\tabulate[rspec={x,0.25,5,2}]
  { \exp x }[x=0][*]
```

⟹

x	$\exp x$
0.0	1.000000
0.2	1.221403
0.4	1.491825
0.6	1.822119
0.8	2.225541

x	$\exp x$
0.00	1.000000
0.25	1.284025
0.50	1.648721
0.75	2.117000
1.00	2.718282

x	$\exp x$
0.00	1.000000
0.25	1.284025
0.50	1.648721
0.75	2.117000
1.00	2.718282

The reason for allowing the inclusion of the `rround` value in `rspec` is because the values included in `rspec`, together with the initial value assigned to the row variable in the vv-list, determine the numbers in the row variable column used to calculate function values. The further settings discussed below in §2.1.3 are *formatting* elements affecting appearance but irrelevant to the actual values calculated.

Table 2.2: Non-uniform row variable specification

key	type	meaning	comment
`rfunc`	token(s)	formula for row var. values	
`rdata`	comma list or macro	list or macro (containing list) of row var. values	
`rfile`	chars	filepath/filename	file contains list of row var. values
`rdelim`	char	item separator for `rdata` or `rfile` lists	defaults to `,` or `;` depending as `.` or `,` is decimal mark
`rverb`	fp (`0`/`0.5`/`1`)	display `rdata`, `rfile` values verbatim (`1`), or slash fractions formatted (`0.5`)	initialized to `0`

2.1.2 Row variable specification: non-uniform case

Occasionally one wants to form a table in which the row variable does not increase or decrease in regular steps; for examples, see *HMF* Tables 1.1 and 3.1. (Tables 9.7, 10.5, 10.10 use two step values and could also be handled with these settings.) For instance, one might want a table of values of simple functions of a list of constants, or a table of function values at π, $\pi/2$, $\pi/3$, $\pi/4$, ..., or at 1, 10, 100, 1000, ..., or a table of function values at thoroughly irregular, perhaps experimentally determined, values.

`numerica-tables` provides two means of specifying such row variables, either by means of a row variable function (`rfunc`), when the row variable values change in a non-uniform but formulaic way, or by explicitly listing the row variable values in a comma list (`rdata`, `rfile`), or with some other separator. In the latter case, the row variable can be displayed either as a sequence of *values*, or verbatim as a sequence of *expressions* – like fractions of π – with the `rverb` setting.

2.1.2.1 `rfunc`

Suppose – perhaps with an interest in the distribution of prime numbers – that we want to create a small table of values of $n/\ln n$ for, say, $n = 10$, 100, 1000, 10000, ... The prospective row variable n is not increasing uniformly, although clearly in a formulaic way. The key `rfunc` is for such cases; in the present instance `rfunc=10^n` where now `n` does increment by a constant amount:

```
\tabulate[rfunc=10^n,rvar=n,rstep=1,rows=7]
  { n/\ln n }[n=1][0] \qquad
\tabulate[rfunc=10^n,rvar=n,rstep=1,rows=7,norules]
  { n/\ln n }[n=1][0]
```

$\Longrightarrow$

n	$n/\ln n$
10	4
100	22
1000	145
10000	1086
100000	8686
1000000	72382
10000000	620421

n	$n/\ln n$
10	4
100	22
1000	145
10000	1086
100000	8686
1000000	72382
10000000	620421

The benefit of the `norules` setting in the second table is obvious.

The variable `n` has two meanings in these tables although only one is publicly visible. Initially `n` is the variable of a step function, incremented by `rstep=1` and taking values `10^n`. Once the table is compiled, `n` denotes these successive function values, $10, 100, \dots, 10000000$. To the *reader*, only this latter meaning is evident, the potentially confusing 'double usage' not so. The initial value `n=1` in the vv-list applies to the row variable function `10^n`, not to the function `n/\ln n` being tabulated (so the error-producing expression `1/\ln 1` does not arise).

2.1.2.2 `rdata`, `rfile`, `rdelim`, `rverb`

A difficulty in reading the last two tables is working out just how many zeros there are in the larger numbers in the left column. They would be more readable 'at a glance' if we could write them in scientific notation. To do that, use the `rdata` and `rverb` keys. In the tables above, `rverb` is absent (corresponding to `rverb=0`); in the tables below `rverb=1`, the effect of which is to use the row variable values in the `rdata` list verbatim. In the first example below, the larger values of the row variable are now expressed in scientific notation, which gives a table that is easier to grasp – but less pleasing to the eye. The right alignment of the column is the culprit, as is seen when presented left-aligned in §2.1.3.2.

```
\tabulate[rdata={10,100,1000,10^4,10^5,10^6,10^7},
    rverb=1,rvar=n]{ n/\ln n }[0] \qquad
\def\mydata{\sfrac14\,\pi,\sfrac13\,\pi,\sfrac12\,\pi,
             \sfrac23\,\pi,\sfrac34\,\pi,\pi}
\tabulate[rdata=\mydata,rverb=1,rvar=k]
  { k }[*]
```

$\Longrightarrow$

n	$n/\ln n$
10	4
100	22
1000	145
10^4	1086
10^5	8686
10^6	72382
10^7	620421

k	k
$^1/_4\,\pi$	0.785398
$^1/_3\,\pi$	1.047198
$^1/_2\,\pi$	1.570796
$^2/_3\,\pi$	2.094395
$^3/_4\,\pi$	2.356194
π	3.141593

In the second example `rverb=1` is used to make a table of simple fractions of π, listed verbatim in the row variable column against their decimal values in the second column (the pointless header row is discussed shortly). When `rverb=1`, `rround` does not apply. The full value is used in the calculation. For the fractions I have used the `\sfrac` command from the `xfrac` package (loaded in the preamble to the present document). `\sfrac` produces an elegant inline fraction with a smaller vertical footprint than `\tfrac` and so is better suited to use in a table column that aligns fraction above fraction. The data has been stored in the macro `\mydata`. By setting `rdata` equal to this macro, the `\tabulate` command gains access to the values stored in it.

In addition to a list or a macro, data for the row variable can also be stored as a list in a file – say `mydata.txt`. If `mydata.txt` is placed in the directory of the current document and `rfile=mydata.txt` entered in the settings option of the `\tabulate` command, the file will be found and the values in the file used for the row variable. Alternatively, the file could be placed in your `texmf` tree and your TEX distro alerted to its presence (by refreshing the filename database). Again `rfile=mydata.txt` in the settings option will ensure the file is found and the contents used for the row variable values. Or, the file could be stored elsewhere and the `rfile` key equated to the full path and filename – something like `rfile=e:/mydocs/mydatafiles/mydata.txt`. This ensures the file will be found and the contents used for the row variable values. Note that even in Windows systems (where file paths use backslashes) the path requires that forward slashes only be used.

The example above uses commas to separate the values accessed by `rdata`. In earlier versions of `numerica-tables` this was the case even if the `comma` package option was used with `numerica` (meaning values using a comma as the decimal mark would need to be enclosed in braces). From version 3.2 of `numerica-tables`, if the `comma` package option is used when calling `numerica`, then the list that `rdata` is equated to (or the file that `rfile` references) should be, by default, a *semicolon* list. It is also possible, from version 3.2, to specify your own delimiter (item separator) by using the `rdelim` setting. For instance, `rdelim=|` would mean that the 'pipe' character separates items in the `rdata` list (or the file referenced by `rfile`).

An irritant with the second table above is the pointless header row. To suppress it I could anticipate and enter `headless` (see §2.4.3) in the settings option. But perhaps a better solution is to present the table horizontally by means of the `transpose` setting. Now the pointless header in the vertical table becomes a pointless first column in the horizontal table. The way to suppress that is by giving a zero value to the setting `rpos` (see §2.1.3.1 for a discussion of `rpos`). `rpos` determines the placement of the row variable column and a zero value suppresses its display entirely. I have also suppressed all rules with the `norules` setting, and reduced the rounding value to 4. At which point the remaining quibble is with the default right alignment of the entries in the header row. I have anticipated again and centred each column with the `calign=c` setting (§2.2.2.6).

```
\def\mydata{\sfrac14\,\pi, \sfrac13\,\pi, \sfrac12\,\pi,
            \sfrac23\,\pi, \sfrac34\,\pi, \pi}
\tabulate[rdata=\mydata,rverb=1,rvar=k,rpos=0,calign=c,
            norules,transpose]{ k }[4*]
```

$\Longrightarrow$

$^1/_4\,\pi$	$^1/_3\,\pi$	$^1/_2\,\pi$	$^2/_3\,\pi$	$^3/_4\,\pi$	π
0.7854	1.0472	1.5708	2.0944	2.3562	3.1416

2.1.2.3 Fraction-form values

The `rverb` setting applies *only* to the `rdata` and `rfile` keys. It has no effect otherwise. Besides `0` and `1`, it can also take a third value, `rverb=1/2`, which can be thought of as 'semi-verbatim'. (`rverb=0.5` or indeed any expression in the syntax of `l3fp` – like `sin(pi/6)` – that evaluates to one half will serve as well.) This renders 'naked' slash fractions from `rdata` or `rfile` input in formatted form using the `\sfrac` command from the `xfrac` package if it is loaded (e.g. $^2/_3$), or in scriptstyle (e.g. $\scriptstyle 2/3$) if it isn't. A 'naked' slash fraction here means the fraction isn't buried within parentheses or other constructs.

I repeat the vertical presentation of the last table but use the 'semi-verbatim' setting. By moving the repeated factor `\pi` from the data file to the function being tabulated, the data file becomes a list of 'naked' slash fractions (and one integer) and the header row becomes meaningful. Note that the integer is displayed as is with the 'semi-verbatim' setting; only the slash fractions are reformatted. Again, I have centred the second column with the `calign=c` setting:

```
\def\mydata{1/4,1/3,1/2,2/3,3/4,1}
\tabulate[rdata=\mydata,rverb=1/2,
          rvar=k,calign=c]
  { k\pi }[k=0][6*]
```

$\Longrightarrow$

k	$k\pi$
$^1/_4$	0.785398
$^1/_3$	1.047198
$^1/_2$	1.570796
$^2/_3$	2.094395
$^3/_4$	2.356194
1	3.141593

2.1.3 Formatting the row variable column & header

The padding option (*) of the trailing optional argument is one way of formatting the row variable values, but to how many decimal places? Aligned left or right or centred? Under what heading – the examples so far have simply used the row variable for the header? And should the row variable column be at the left of the table, or the right – or both? These and related questions are answered by assigning values to the keys listed in Table 2.3.

Table 2.3: Formatting the row variable column & header

key	type	meaning	initial
`rpos`	int (`0`...`4`)	column placement	`1`
`ralign`	char (`r`/`c`/`l`)	horizontal alignment	`r`
`rfont`	chars	font (`\math<chars>`)	
`rhead`	tokens	header	
`rhnudge`	fp	nudge header `<fp>` mu	`0`
`rhsize`	int (`-4`...`5`)	font size relative to `0=\normalsize`	`0`
`rmath`	char (`s`/`t`/`d`)	script-, text-, displaystyle	`t`
`rvar'`	tokens	2nd row variable col. spec.	
`rhead'`	tokens	header of 2nd row var. col. (if it exists)	
`rhnudge'`	fp	nudge 2nd row var. col. header `<fp>` mu	`0`

2.1.3.1 Position in the table: `rpos`

By default, the row variable column is the *first* column of the table. Its position is determined by the value of the key `rpos`:

- `rpos=0`, suppressed (no row variable column);
- `rpos=1`, first column (the default);
- `rpos=2`, last column;
- `rpos=3`, first and last columns (e.g. see §2.3);
- `rpos=4`, first column and a last column with values produced by a user-defined function of the first; see §2.4.6;
- Any other integer acts like `rpos=1`.

When a table is transposed, these settings apply to the *new* row variable column (the old header row). For instance, with `rpos=2` we get what was originally the header row treated as the row variable column and placed on the right:

```
\tabulate[rspec={x,0.2,5},rpos=2,transpose,norules]
    { \tan x }[x=0.2][*]
```

$\Longrightarrow$

0.2	0.4	0.6	0.8	1.0	x
0.202710	0.422793	0.684137	1.029639	1.557408	$\tan x$

Adjoined multi-function tables By assigning different values to `rpos` it is possible to include multiple functions in the same table by adjoining distinct single column tables. *HMF* has many, many examples where multiple functions (like the trigonometric or hyperbolic functions) are tabulated in separate columns of the same table.

`numerica-tables` has a systematic way of doing this, described in §2.3, but with the settings discussed to this point, the same effect can be achieved by the present more naive means. The example below displays as a single multi-columned table but is composed of three separate tables. I have used three different `rpos` settings (`rpos=1`, the default setting, is implicit in the first) to achieve this and ended all but the last `\tabulate` command with the LaTeX comment character `%`. These characters are essential if the tables are to abut exactly. Omitting them results in a space between the tables.

```
\tabulate[rspec={x,0.2,6}]
  { \sin x }[x=0][*]%
\tabulate[rpos=0,rspec={x,0.2,6}]
  { \cos x }[x=0][*]%
\tabulate[rpos=2,rspec={x,0.2,6}]
  { \tan x }[x=0][*]
```

⟹

x	$\sin x$	$\cos x$	$\tan x$	x
0.0	0.000000	1.000000	0.000000	0.0
0.2	0.198669	0.980067	0.202710	0.2
0.4	0.389418	0.921061	0.422793	0.4
0.6	0.564642	0.825336	0.684137	0.6
0.8	0.717356	0.696707	1.029639	0.8
1.0	0.841471	0.540302	1.557408	1.0

It is possible to similarly join transposed tables (you need the `headless` setting), stacked one below the other, but again that effect can be achieved more readily in a systematic way – see §2.3.

2.1.3.2 Alignment: `ralign`

By default, the alignment of all columns including the row variable column is to the right, as in the examples so far. This lends itself to clean output when padding with zeros is activated (the `*` in the trailing argument) or when some values are negative, since minus signs can interfere with alignment of digits in left or centred alignments. In the first example below I present an earlier table where the row variable values were a mix of powers of 10 multiplied out and in exponent form. In that table (see §2.1.2.2) with its right alignment the eye stumbled over the exponents. In left alignment, it is a column of '1's which is aligned and the alignment is 'smooth'. Left alignment is achieved by setting `ralign=l` (a lowercase 'L'). In the second table I have chosen a centred alignment for the row variable column, `ralign=c` (and once more used `calign=c` for the function-value column). The other possible setting for `ralign` is the default right alignment, `ralign=r`.

```
\tabulate[rdata={10,100,1000,10^4,10^5,10^6,10^7},
    rverb=1,rvar=n,ralign=l]
  { n/\ln n }[0]\qquad
 \tabulate[rspec={x,0.01,7},rround=2,ralign=c,calign=c]
   { \sin x }[x=1.51][*]
```

$\Longrightarrow$

n	$n/\ln n$
10	4
100	22
1000	145
10^4	1086
10^5	8686
10^6	72382
10^7	620421

x	$\sin x$
1.51	0.998152
1.52	0.998710
1.53	0.999168
1.54	0.999526
1.55	0.999784
1.56	0.999942
1.57	1.000000

2.1.3.3 Font: `rfont`

You may wish to distinguish the values in the row variable column from the values in the function-value column typographically, for instance by bolding them. This is effected through the setting `rfont`. Possible values for this setting are `rm` (roman), `bf` (bold face), `it` (italic), `sf` (sans serif), and `tt` (typewriter). For instance, `rfont=bf` applies `\mathbf` to each row variable value, `rfont=tt` applies `\mathtt` to each row variable value, and so on. If you set `rfont=xx` when `\mathxx` has not been defined, then a LaTeX error will result. Equally, something like `rfont=cal` (where `\mathcal` *has* been defined) should not cause error but will give unexpected results if the digits 0 to 9 are not part of the font. In the next example, I repeat the last example in 'vertical' form but with the row variable column values bolded to draw attention to their placement on the right – but note that the `rfont` setting has no effect on the row variable header.

```
\tabulate[rspec={x,0.2,5},rpos=2,rfont=bf]
    { \tan x }[x=0.2][*] \qquad
\tabulate[rspec={x,0.2,5},rpos=2,rfont=bf,
            rhead=\boldsymbol{x}\phantom{.}]
    { \tan x }[x=-0.4][*]
```

$\Longrightarrow$

$\tan x$	x
0.202710	**0.2**
0.422793	**0.4**
0.684137	**0.6**
1.029639	**0.8**
1.557408	**1.0**

$\tan x$	$\boldsymbol{x}$
-0.422793	$-\mathbf{0.4}$
-0.202710	$-\mathbf{0.2}$
0.000000	**0.0**
0.202710	**0.2**
0.422793	**0.4**

2.1.3.4 Row variable header: `rhead`

The default header is the row variable symbol. In the second table that has been changed by giving a value to the key `rhead`. I have used `rhead=\boldsymbol{x}` (rather than `\mathbf{x}`) in order to get an italicized bold symbol. I've also added a `\phantom{.}` to `rhead` in order to push the (by default) right-aligned header leftwards and give a more centred appearance to its positioning. Math delimiters (`$ $`) are inserted automatically by `numerica-tables`. If you do not want any header at all for the row variable column, entering `rhead=` , an empty setting, will achieve this – at least from version 3.2 of `numerica-tables`. In earlier versions you needed to set `rhead` to (e.g.) a space of some kind like `\ ` (an inter-word space).

2.1.3.5 Nudging the header: `rhnudge`

Choosing a centred alignment for the header to the row variable column will generally not be appropriate when negative values are involved. In an earlier table I used a phantom to get around this. Another way which avoids obscuring the true content of the `rhead` setting with positioning commands is to use the setting `rhnudge`.

Both tables below have the default right alignment of the row variable column and use `rhnudge` to adjust the header position. For positive values, `rhnudge` works in the *opposite* sense to the alignment, to the left for right alignment, and to the right otherwise. The units for nudging are mu (math units, 18 to a quad), but only a number should be specified; the 'mu' is supplied by `numerica-tables`.

```
\tabulate[rvar=x,rstep=0.25,rstop=0.5,rround=2,
          rfont=bf,rhead=\boldsymbol{x},rhnudge=9]
  { \sin x }[x=-0.5][4*]\qquad
\tabulate[rvar=x_{\text{int}},rstep=1,rstop=4,
          rround=0,rfont=bf,rhnudge=-12,
          rhead=\boldsymbol{x_{\text{int}}}]
  { \exp x_{\text{int}} }[x_{\text{int}}=0][4*]
```

$\Longrightarrow$

$\boldsymbol{x}$	$\sin x$
−0.50	−0.4794
−0.25	−0.2474
0.00	0.0000
0.25	0.2474
0.50	0.4794

$\boldsymbol{x}_{\text{int}}$	$\exp x_{\text{int}}$
0	1.0000
1	2.7183
2	7.3891
3	20.0855
4	54.5982

In the second table the row variable takes single digit integer values, while the row variable name occupies more than one character. With a right alignment the header would protrude out to the left. By giving `rhnudge` a *negative* value (`rhnudge=-12` in the example) it is brought back to a centred position in the row variable column.

2.1.3.6 Font size of the header: `rhsize`

The font size of the header can be changed by assigning an integer value to the key `rhsize`. Valid values range from -4, corresponding to `\tiny`, through 0, corresponding to `\normalsize`, to 5, corresponding to `\Huge`. The actual values in points differ depending as `\normalsize` is 10pt, 11pt, or 12pt, but range from 5pt for `\tiny` when `\normalsize` is 10pt, to 24.88pt for `\Huge` in all three cases. See also §2.2.2.9.

2.1.3.7 `rmath`

By default the row variable column is displayed in textstyle (`\textstyle`). By setting `rmath` to `s`, `t` or `d` it is possible to explicitly choose `\scriptstyle`, `\textstyle` or `\displaystyle` for its display. `rmath` is relevant when one might want to display more complicated structures than numbers in the row variable column, e.g. verbatim items when using `rdata` or `rfile`, or in transposed multi-function tables; see the example in §2.3. Returning to an earlier example, suppose the macro `\mydata` contains a list of fractions, not in slash form but `\frac`-form:

```
\def\mydata{\frac14,\frac13,\frac12,\frac23,\frac34,1}
\tabulate[rdata=\mydata,rverb=1,rvar=k,calign=c]
  { k\pi }[k=0][6*]\qquad
\renewcommand\arraystretch{1.2}
\tabulate[rdata=\mydata,rverb=1,rvar=k,calign=c]
  { k\pi }[k=0][6*]\qquad
\renewcommand\arraystretch{1}
\tabulate[rdata=\mydata,rverb=1,rvar=k,rmath=s,calign=c]
  { k\pi }[k=0][6*]
```

$\Longrightarrow$

k	$k\pi$
$\frac{1}{4}$	0.785398
$\frac{1}{3}$	1.047198
$\frac{1}{2}$	1.570796
$\frac{2}{3}$	2.094395
$\frac{3}{4}$	2.356194
1	3.141593

k	$k\pi$
$\frac{1}{4}$	0.785398
$\frac{1}{3}$	1.047198
$\frac{1}{2}$	1.570796
$\frac{2}{3}$	2.094395
$\frac{3}{4}$	2.356194
1	3.141593

k	$k\pi$
$\scriptstyle\frac{1}{4}$	0.785398
$\scriptstyle\frac{1}{3}$	1.047198
$\scriptstyle\frac{1}{2}$	1.570796
$\scriptstyle\frac{2}{3}$	2.094395
$\scriptstyle\frac{3}{4}$	2.356194
$\scriptstyle 1$	3.141593

The first table is unacceptable; the second (with `\arraystretch` set to 1.2) and the third (with `rmath=s`) are at least readable but neither is really satisfactory. Better is to use `\sfrac`, either directly or by the ‘semi-verbatim’ `rverb` setting discussed earlier in §2.1.2.3.

2.1.3.8 `rvar’`, `rhead’`, `rhnudge’`

These settings become relevant only when `rpos=4`; see §2.4.6.

Table 2.4: Column-variable specification

key	type	meaning	comment
`cvar`	token(s)	column variable	
`cstep`	real num.	step size	
`cstop`	real num.	stop value	either `cstop`
`cols`	int	number of columns	or `cols`
`cspec`	comma list	`{cvar,cstep,cols}`	short form spec.
`chround`	int	col. var. rounding	default: `0`

2.2 Column-variable settings

When a function of *two* variables is tabulated, we generally think of one variable as the primary variable and the other as a parameter. To tabulate such a function, we use the primary variable as the row variable and create a succession of adjacent columns of function values according to the different values of the parameter. In this document I call this second variable the *column* variable (`cvar`). Just as with the row variable, to create a table we need its start value (specified in the vv-list), and its step (`cstep`) and stop (`cstop`) values.

```
\tabulate[rspec={x,0.2,5},cvar=k,cstep=2,cstop=9]
  { \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

$x\backslash k$	3	5	7	9
0.2	0.5646	0.8415	0.9854	0.9738
0.4	0.9320	0.9093	0.3350	−0.4425
0.6	0.9738	0.1411	−0.8716	−0.7728
0.8	0.6755	−0.7568	−0.6313	0.7937
1.0	0.1411	−0.9589	0.6570	0.4121

In this example `cvar=k` is the column variable. I have chosen a step size `cstep=2` and a stop value `cstop=9`. The start value (`k=3`) is specified in the vv-list. Although in the example these values are numbers, all three values could be LaTeX expressions that evaluate to numbers. In particular, the expressions for step and stop values may include the row and column variables (in the example x and k) which are assigned their *initial* vv-list values.

`numerica-tables` has automatically inserted the row variable header which now has a double function: it serves also as the 'header' of the header row. As you can see $x\backslash k$ has been inserted into this cell. From version 3.2 of `numerica-tables` the automatic insertion of this form of header in a multi-column table occurs if `rhead` is omitted. (If you want nothing to show, put `rhead=` , an empty setting.) This form of header above the row variable column occurs throughout *HMF*. In the present example it shows the reader that the numerical values displayed in the row variable column are values of x and the values in the column headers are values of k.

As with the row variable, rather than using a stop value, `cstop`, you can specify the number of columns, `cols`, explicitly. I could have replaced `cstop=9` with `cols=4` to get the same result. `cols` specifies the number of *function-value* columns; the row variable column is ignored for this count. A quirk of using `cols` is that it is then possible to have a *zero* step size, `cstep=0`. (A similar comment applies to `rows` and `rstep`.) For an instance where this is useful, see the last example in §2.2.2.7.

Again as with the row variable, the column specification can be condensed into a comma list with the key `cspec`. This is a 3-element comma list of the form `{cvar,cstep,cols}` (and as for `rspec`, with an optional fourth element, `chround`; see §2.2.1). Thus, for the preceding table I could also have written

```
\tabulate[rspec={x,0.2,5},cvar=k,cstep=2,cols=4]
  { \sin kx }[k=3,x=0.2][4*]
```

or more succinctly

```
\tabulate[rspec={x,0.2,5},cspec={k,2,4}]
  { \sin kx }[k=3,x=0.2][4*]
```

and produced the same table.

Like their row equivalents, `cstep`, `cstop` and `cols` can all be LaTeX expressions, and like those equivalents, the first two are evaluated *after* the vv-list and so may depend not only on numbers and constants but also on the *initial* values of the row and column variables, which are assigned in the vv-list. `cols` is evaluated *before* the vv-list; it may be a LaTeX expression but cannot depend on the row or column variable.

Multi-column tables can be transposed: simply add `transpose` to the settings option. I have done this for the last example. Transposition automatically changes the positions of k and x in the new row variable header:

```
\tabulate[rspec={x,0.2,5},cvar=k,cstep=2,cstop=9,
        transpose]{ \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

$k\backslash x$	0.2	0.4	0.6	0.8	1.0
3	0.5646	0.9320	0.9738	0.6755	0.1411
5	0.8415	0.9093	0.1411	−0.7568	−0.9589
7	0.9854	0.3350	−0.8716	−0.6313	0.6570
9	0.9738	−0.4425	−0.7728	0.7937	0.4121

To my eye, this table looks odd. Admittedly, the oddness is exaggerated by the new row variable column being a column of integers; `ralign=c` would help there. But the underlying problem remains: we expect the coarser graining in the column variable and its header row, and the finer graining in the row variable and its column. Transposition has confounded this expectation.

2.2.1 Rounding: `chround`

Generally the column variable increments in larger steps than the row variable. Often these are integer steps. For that reason the default rounding value of the column variable is 0 (as against 1 for the row variable), but it can be changed by assigning a different value to the key `chround`.

For transposed tables, rounding is applied *before* transposition. The function values displayed in the body of the table are calculated for the *displayed* values of the row and column variables. Transposition doesn't change the displayed function values and therefore shouldn't change the displayed row and column variable values, irrespective of their placement in the table. In the last (transposed) example the 'new' column headers show that the original row variable rounding has been retained. As an essential part of the column variable specification, `chround` can be included as an optional fourth element in `cspec`.

Since the default column variable rounding is 0, any non-integer increment to that variable needs an explicit value to be assigned to `chround`. For instance if k increments by, say, 0.25, then `chround=2` will need to be entered in the settings option of the `\tabulate` command, or `2` entered as the fourth element of `cspec`:

```
\tabulate[rspec={x,0.2,5},ralign=c,
          cspec={k,0.25,4,2}]
  { \sin kx }[k=3,x=0.2][*4]
```

$\Longrightarrow$

$x\backslash k$	3.00	3.25	3.50	3.75
0.2	0.5646	0.6052	0.6442	0.6816
0.4	0.9320	0.9636	0.9854	0.9975
0.6	0.9738	0.9290	0.8632	0.7781
0.8	0.6755	0.5155	0.3350	0.1411
1.0	0.1411	−0.1082	−0.3508	−0.5716

2.2.2 Column header formatting

Column header formatting depends on whether the table has a single column of function values, or multiple columns of function values. For the latter there are a number of built-in style settings for the header, accessed by assigning a value (`0`, `1`, `2`, `3`) to the setting `chstyle`. This can (sometimes) be adjusted with the settings `chfont`, `chnudge` and `chmath`. If these built-in styles don't satisfy then it is possible to define your own header to the function-value columns by using the setting `chead`.

2.2.2.1 Single-column header

When there is only one column of function values, the function being tabulated is by default set as the header to the column. This can be nudged left or right by giving a numerical value to `chnudge`. It can be set in script-, text- or displaystyle

Table 2.5: Formatting the column variable header

key	type	meaning	default
`chstyle`	int (`0`...`4`)	header style	`0`
`chead`	token(s)	user-defined col. var. header	
`calign`	char (`r`/`c`/`l`)	column alignment	`r`
`chnudge`	fp	nudge header `chnudge` mu	`0`
`chfont`	chars	font (`\math<chars>`)	
`chmath`	char (`s`/`t`/`d`)	script-, text- or displaystyle	`t`
`chsize`	int (`-4`...`5`)	font size relative to `0 => \normalsize`	`0`

(see the example) by setting `chmath` to `s`, `t` (the default) or `d`. `chfont`, however, has no effect in the single column case.

```
\tabulate[rspec={x,0.25,5},rround=2,chmath=s]
   { \frac{2}{\sqrt\pi} e^{-x^2} }[x=0][*] \qquad
\tabulate[rspec={x,0.25,5},rround=2]
   { \frac{2}{\sqrt\pi} e^{-x^2} }[x=1][*] \qquad
\tabulate[rspec={x,0.25,5},rround=2,chmath=d]
   { \frac{2}{\sqrt\pi} e^{-x^2} }[x=2][*]
```

$\Longrightarrow$

x	$\scriptstyle\frac{2}{\sqrt{\pi}}e^{-x^2}$
0.00	1.128379
0.25	1.060014
0.50	0.878783
0.75	0.642931
1.00	0.415107

x	$\frac{2}{\sqrt{\pi}}e^{-x^2}$
1.00	0.415107
1.25	0.236521
1.50	0.118930
1.75	0.052775
2.00	0.020667

x	$\dfrac{2}{\sqrt{\pi}}e^{-x^2}$
2.00	0.020667
2.25	0.007142
2.50	0.002178
2.75	0.000586
3.00	0.000139

You may want some other header to the function-value column, one of your own choosing. In that case give `chead` a value. You are responsible for the entire content, including any math environment; see the examples at §2.5.1.1.

2.2.2.2 Multi-column header

The default header style lists the column variable *value* at the head of each function-value column. Implicitly it corresponds to `chstyle=0`.

```
\tabulate[rspec={x,0.2,5},cvar=k,cstep=2,cstop=9]
  { \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

$x\backslash k$	3	5	7	9
0.2	0.5646	0.8415	0.9854	0.9738
0.4	0.9320	0.9093	0.3350	−0.4425
0.6	0.9738	0.1411	−0.8716	−0.7728
0.8	0.6755	−0.7568	−0.6313	0.7937
1.0	0.1411	−0.9589	0.6570	0.4121

Unless `rhead` is explicitly specified, `numerica-tables` will automatically insert a row variable header, like $x\backslash k$ in the example, where the backslash separates row from column variable. (But see also §2.2.2.3 below.) *HMF* contains a multitude of instances of this style; see Tables 9.7, 17.5, 21.1, 24.3, 27.4, etc. for examples. As noted, you can override the default assignment to `rhead` with your own specification, but for `chstyle > 0` this is no longer true.

2.2.2.3 `diagbox`, `slashbox`

Rather than having $x\backslash k$ or something similar automatically inserted, you might be tempted to use `\diagbox` from the `diagbox` package, or `\backslashbox` from the earlier `slashbox` package, say `rhead=\diagbox{x}{k}`. Because `rhead` wraps its contents in math delimiters (`$` signs) this causes a LaTeX error. The solution is to *add* `$` signs: `rhead=$\diagbox{x}{k}$` or, since $x$ and $k$ should be in math-italic, `rhead=$\diagbox{$x$}{$k$}$`. You might therefore try

```
\tabulate[rspec={x,0.2,5},ralign=c,
            cvar=k,cstep=2,cstop=9,
            rhead=$\diagbox{$x$}{$k$}$]
  { \sin kx }[k=3,x=0.2][4*]
```

but the diagonal box is far too large. One solution is to use the optional first argument of the `\diagbox` command. In that argument one can specify an explicit `height` and `width` but I prefer to work solely with the settings available in `numerica-tables`. In that spirit, put `rhsize=-4`, its smallest value. Alas this also shrinks the displayed size of the variables $x$ and $k$. To counteract that, explicitly specify `\small$x$` and `\small$k$` for the mandatory `\diagbox` arguments, the `\small` overriding the `rhsize` setting to give a presentable result:

```
\tabulate[rspec={x,0.2,5},ralign=c,rhsize=-4,cspec={k,2,4},
            rhead=$\diagbox{\small$x$}{\small$k$}$]
  { \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

$k$ $x$	3	5	7	9
0.2	0.5646	0.8415	0.9854	0.9738
0.4	0.9320	0.9093	0.3350	−0.4425
0.6	0.9738	0.1411	−0.8716	−0.7728
0.8	0.6755	−0.7568	−0.6313	0.7937
1.0	0.1411	−0.9589	0.6570	0.4121

However, the `booktabs` manual admonishes against the use of vertical rules and double rules and generally advises to use as few rules as possible. By adding a line (admittedly *diagonal*, so only half sinning) the `diagbox` or `slashbox` packages are working in the opposite direction, more suited to tables in the older 'grid' style. In the present document I shall use $x\backslash k$ and make no further use of either `\diagbox` or `\backslashbox`.

2.2.2.4 Built-in styles: `chstyle`

There are some built-in header styles, depending on the value of the setting `chstyle`. No value for this setting corresponds to `chstyle=0`, which results in column variable values at the heads of function-value columns and something like $x\backslash k$ at the head of the row variable column, as previously described.

`chstyle=1` changes the header of the *first* function-value column to the form *variable=value*, e.g. $k = 3$ in the example below. This may be apt when a small rounding value is being used and the resulting columns are narrow. I can find only one real instance in *HMF*, Table 26.7. Note that the row variable header, inserted automatically, simplifies in this case to x rather than $x\backslash k$.

```
\tabulate[rspec={x,0.2,5},cspec={k,2,3},chstyle=1]
  { \sin kx }[k=3,x=0.2][3*]
```

$\Longrightarrow$

x	$k = 3$	5	7
0.2	0.565	0.841	0.985
0.4	0.932	0.909	0.335
0.6	0.974	0.141	−0.872
0.8	0.675	−0.757	−0.631
1.0	0.141	−0.959	0.657

`chstyle=2` changes the header of all function-value columns to the form *variable=value*. In *HMF* examples are Tables 7.4, 7.9, 10.10, 16.6, etc. Again, the row variable header is inserted automatically and displays the row variable.

```
\tabulate[rspec={x,0.2,5},cspec={k,2,3},chstyle=2]
  { \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

x	$k = 3$	$k = 5$	$k = 7$
0.2	0.5646	0.8415	0.9854
0.4	0.9320	0.9093	0.3350
0.6	0.9738	0.1411	−0.8716
0.8	0.6755	−0.7568	−0.6313
1.0	0.1411	−0.9589	0.6570

`chstyle=3` fills each column variable header with the expression being tabulated but with the column variable replaced by its respective values. See *HMF* Tables 5.4, 8.1, 9.1, 19.1, etc. for examples.

```
\tabulate[rspec={x,0.2,5},cspec={k,2,3},chstyle=3]
  { \sin kx }[k=1,x=0.2][4*]
```

$\Longrightarrow$

x	$\sin 1x$	$\sin 3x$	$\sin 5x$
0.2	0.1987	0.5646	0.8415
0.4	0.3894	0.9320	0.9093
0.6	0.5646	0.9738	0.1411
0.8	0.7174	0.6755	−0.7568
1.0	0.8415	0.1411	−0.9589

Notable here is that the column variable takes the value 1, and the 1 is displayed ($\sin 1x$) where it would normally be suppressed. A similar situation can arise when the column variable takes the value 0. The legacy setting `chstyle=4` will, in the present instance, display $\sin 1x$ as $\sin x$ but is deprecated – given the variety of ways that 1 (and 0) can appear in formulas, it seems more straightforward in to set `chstyle=0` (or omit it) and …

2.2.2.5 User-defined header: `chead`

… form `chead` oneself, with the relevant math delimiters (`$`) and tabbing ampersands (`&`). (Always, to use `chead`, `chstyle` must be omitted or set to zero.) In the example,

```
chead=$\sin x$&$\sin3x$&$\sin5x$
```

would do. It is a header for the function-value columns; a header for the row variable column is not included. Note that there is also no end of row mark which is inserted automatically.

Another way to tackle the particular issue in the present case is to form a multi-function table (see §2.3) of, say, $\sin x$, $\sin 3x$, $\sin 5x$.

2.2.2.6 Alignment: `calign`

By default the function-value columns are aligned right, `calign=r`. Also available are `calign=c` for centred alignment and `calign=l` (lowercase L) for left alignment. A table containing both positive and negative values is best set with right alignment and number padding (`*` in the trailing optional argument) to avoid the misalignment of digits in columns caused by the presence of minus signs. (Handling signs in tables is discussed later; see §2.5.2.2.) Varying numbers of digits *before* the decimal point may be best handled with the t-notation of §2.5.2. Examples where a centred alignment is helpful, centring the header above uniform columns of figures (same number of digits before and after the decimal point) have already been given; see §2.1.2.3 and §2.1.3.2.

2.2.2.7 Nudging header entries: `chnudge`

As with the row variable header, it is possible to nudge the column headers to left or right. A positive nudge value shifts the header in the *opposite* sense to the alignment by the specified number of mu (math units; 18 to a quad), to the left in a right alignment and to the right otherwise. In version 3.2 of `numerica-tables,` the key `chnudge` is a comma list of nudge values (as against a single value in earlier versions). As with `rhnudge`, only numbers should be supplied; `numerica-tables` inserts the unit (`mu`). Often a single value is all that is required. For instance, in the next example `chnudge=9` suffices to nudge the column headers to the left but leave the function values (with their potentially awkward minus signs) right aligned.

```
\tabulate[rspec={x,0.2,5},ralign=c,
          cspec={k,2,3},chstyle=2,chnudge=9]
  { \sin kx }[k=3,x=0.2][*]
```

$\Longrightarrow$

x	$k=3$	$k=5$	$k=7$
0.2	0.564642	0.841471	0.985450
0.4	0.932039	0.909297	0.334988
0.6	0.973848	0.141120	−0.871576
0.8	0.675463	−0.756802	−0.631267
1.0	0.141120	−0.958924	0.656987

But if the values of k in this example had different numbers of digits, no single `chnudge` value would do. In that case give `chnudge` a comma-list of values, potentially one for each function-value column. If the comma list has fewer entries than the number of columns, the last entry is used for all ‘left over’ columns. For instance, in the following example, `chnudge` is given three values, the third of which is also used for the fourth function-value column:

```
\tabulate[rspec={x,0.2,5},ralign=c,cspec={k,50,3},
          chstyle=2,chnudge={13.5,9,4.5}]
  { \cos kx }[k=1,x=0.2][*]
```

$\Longrightarrow$

x	$k=1$	$k=51$	$k=101$	$k=151$
0.2	0.980067	−0.714266	0.218573	0.347468
0.4	0.921061	0.020351	−0.904451	−0.758532
0.6	0.825336	0.685194	−0.613951	−0.874600
0.8	0.696707	−0.999172	0.636065	0.150740
1.0	0.540302	0.742154	0.892005	0.979355

Negative nudges can be useful when a column header is longer than the displayed function values. Just for the pleasure of providing an instance where a *zero* step value is useful, the following small table allows an immediate comparison of a negative nudge with no nudge. I have adjusted the separation of the columns with `\tabcolsep` to provide a more adequate separation of the headers:

```
\setlength\tabcolsep{9pt}
\tabulate[rspec={n,100,5},ralign=c,rround=0,
          cspec={k,0,2},chstyle=3,chnudge={-21,0}]
```

```
  { (1+k/n)^n }[k=1,n=100][3*]
\setlength\tabcolsep{6pt}
```

$\Longrightarrow$

n	$(1+1/n)^n$	$(1+1/n)^n$
100	2.705	2.705
200	2.712	2.712
300	2.714	2.714
400	2.715	2.715
500	2.716	2.716

2.2.2.8 Math style: `chmath`

The `chmath` setting is a comma list of the values `s`, `t`, or `d` presenting the header of the corresponding function-value column in either scriptstyle, textstyle or displaystyle respectively. The setting is initialized to `t`. If there are more function-value columns in the table than entries in `chmath` the last value in `chmath` is used for the extra columns. See the first table in §2.3 (on multi-function tables) for an example of this setting's use.

2.2.2.9 Font size: `chsize`

The font size of the header row entries can be changed with the setting `chsize` which is a comma list of integer values, each integer in the range -4 to 5 corresponding to a LaTeX font size command according to the scheme

-4	`\tiny`	1	`\large`
-3	`\scriptsize`	2	`\Large`
-2	`\footnotesize`	3	`\LARGE`
-1	`\small`	4	`\huge`
0	`\normalsize`	5	`\Huge`

Should there be more columns in the table than entries in `chsize`, then the last value in `chsize` is used for the extra columns. In the following example, the font size has been tweaked by means of the setting `chsize={1,-1,1}` to bring the character size of the column headings into greater conformity:

```
\tabulate[ff,rspec={x,0.5,4},chsize={1,-1,1}]
  { \frac{ e^{2x}-1}{ e^{2x}+1},\tanh x,
    \frac{ \sinh x}{\cosh x}} [x=0.5] [4*]
```

$\Longrightarrow$

x	$\frac{e^{2x}-1}{e^{2x}+1}$	$\tanh x$	$\frac{\sinh x}{\cosh x}$
0.5	0.4621	0.4621	0.4621
1.0	0.7616	0.7616	0.7616
1.5	0.9051	0.9051	0.9051
2.0	0.9640	0.9640	0.9640

2.3 Multiple functions in a single table

As noted in §2.1.3.1, using adjoining tables in order to tabulate more than one function at a time is 'clunky'. Much better is to enter the functions in the main argument of a `\tabulate` command separated by a specified mark, then alert `\tabulate` that this has happened by setting `numerica`'s `ff` key.

The default multi-function delimiter is the comma if the decimal point is a dot (period), or the semicolon if it is a comma (`numerica` loaded with the `comma` package option). With the default delimiter it suffices to enter `ff` in the settings option; otherwise enter `ff=<mark>` there. For example `ff=|` makes the 'pipe' character the multi-formula delimiter. If the `ff` key is overlooked then multiple formulas in the main argument will almost certainly cause a LaTeX error.

The following example (of the so-called 'Einstein functions') illustrates both a multi-function table using the default comma as separator, and the use of `chmath` to shrink the function in the last column. If not shrunk, $\ln(1-e^{-x})$ appears too large compared to the symbols in the fractions in the middle columns:

```
\tabulate[ff,rspec={x,0.3,4},rround=2,rhnudge=9,
          chnudge={0,9,0},chmath={t,t,s}]
  { \frac{x^2e^x}{(e^x-1)^2}, \frac{x}{e^x-1},
      \ln(1-e^{-x}) }[x=0.15][5*]
```

$\Longrightarrow$

x	$\frac{x^2e^x}{(e^x-1)^2}$	$\frac{x}{e^x-1}$	$\ln(1-e^{-x})$
0.15	0.99813	0.92687	−1.97118
0.45	0.98329	0.79182	−1.01508
0.75	0.95441	0.67144	−0.63935
1.05	0.91298	0.56523	−0.43069

In the next example I transpose this table, not because it is a good example of a table that might benefit from this, but because it illustrates a problem that can arise when a multi-function table is transposed. On transposition the current header row becomes the row variable column:

```
\tabulate[ff,rspec={x,0.3,4},rround=2,ralign=l,
          chnudge={0,9},rules=hB,chstyle=1,transpose]
  { \frac{x^2e^x}{(e^x-1)^2}, \frac{x}{e^x-1},
       \ln(1-e^{-x}) }[x=0.15][5*]
```

$\Longrightarrow$

	$x=0.15$	0.45	0.75	1.05
$\frac{x^2e^x}{(e^x-1)^2}$	0.99813	0.98329	0.95441	0.91298
$\frac{x}{e^x-1}$	0.92687	0.79182	0.67144	0.56523
$\ln(1-e^{-x})$	−1.97118	−1.01508	−0.63935	−0.43069

What is immediately noticeable is the disparity in size of the characters forming $\ln(1-e^{-x})$ compared to those in the fractions above it. This was the reason for the `chmath={t,t,s}` setting in the original table, but `rmath` is a single value applying to the *whole* row variable column, so that solution is not available.

Two possibilities come to mind. One is to force the function being tabulated into scriptstyle by tabulating

```
\vphantom{\frac11}\scriptstyle\ln(1-e^{-x})
```

(The phantom ensures the spacing between the rows is the same.) This new function produces exactly the same values as the previous one did. The other possibility is to set `rmath=d`, forcing all members of the row variable column, in particular the fractions, into displaystyle. With this setting the `\scriptstyle` can be omitted (but the phantom retained).

Another example, now using `ff=|` as the function separator, places the row variable column on both sides of the table (`rpos=3`) and uses the `o` setting (see `numerica.pdf`) to indicate that arguments of the trig functions are in degrees:

```
\tabulate[ff=|,o,rpos=3,rround=0,chnudge=9,
          rvar=\theta,rstep=15,rstop=90]
  { \sin \theta | \cos \theta }[\theta=0][*]
```

$\Longrightarrow$

θ	$\sin\theta$	$\cos\theta$	θ
0	0.000000	1.000000	0
15	0.258819	0.965926	15
30	0.500000	0.866025	30
45	0.707107	0.707107	45
60	0.866025	0.500000	60
75	0.965926	0.258819	75
90	1.000000	0.000000	90

The table suggests a space saving possibility: since sin and cos are complementary functions ($\sin(90-\theta)=\cos\theta$), values in the bottom half of the table duplicate values in the top half, only with the columns reversed. This is the reason for the `rpos=4` setting discussed in §2.4.6, which enables complementary functions to be tabulated in 'half tables'.

2.4 Whole-of-table formatting

There are a number of settings that affect the appearance of the table as a whole, things like the position of the row variable column, the grouping of function values in a column into blocks to aid readability, the presence of horizontal rules, the use of a collective column title, or of a footer row, or display of the table 'horizontally' in rows rather than columns. I discuss these here (Table 2.6).

2.4.1 Title for function-value columns: `ctitle`

The function-value columns have individual headings, formatted in the various ways provided by the settings already discussed, but it can also be helpful to have a collective title for these columns. For instance our table of Einstein functions in §2.3 would have benefited from this. The need is met with the

Table 2.6: Table formatting

key	type	meaning	initial
`ctitle`	token(s)	collective title for columns	
`csubttl`	token(s)	subtitle row for function-value cols	
`calign`	char(`r`/`c`/`l`)	column alignment	`r`
`headless`		suppress header row	
`foot`	token(s)	table-wide footer row	
`rules`	char(s)	horizontal rule spec.	`ThB`
`norules`		cancel all rules	
`rpos`	int (`0`...`4`)	row variable col. position(s)	`1`
`rbloc`	integer comma list	row block specification	
`rblocsep`	length	extra space between row blocks	`1ex`
`transpose`		show funct. vals in rows	
`valign`	char (`t`/`m`/`b`)	vertical alignment of table relative to text baseline	`m`

`ctitle` key. This can be set to whatever you like (e.g. `ctitle=Fred`), taking care to shield commas with braces. By default, the title is set between math delimiters (`$` signs). For that reason in the example I have placed it inside a `\text` command:

```
\tabulate[ff,rspec={x,0.15,6},rround=2,rhnudge=9,
          chnudge={0,9,0},chmath={t,t,s},
          ctitle=\text{Einstein functions}]
  { \frac{x^2e^x}{(e^x-1)^2}, \frac{x}{e^x-1},
      \ln(1-e^{-x}) }[x=0.15][5*]
```

		Einstein functions		
	x	$\frac{x^2e^x}{(e^x-1)^2}$	$\frac{x}{e^x-1}$	$\ln(1-e^{-x})$
	0.15	0.99813	0.92687	−1.97118
$\Longrightarrow$	0.30	0.99253	0.85749	−1.35023
	0.45	0.98329	0.79182	−1.01508
	0.60	0.97053	0.72982	−0.79587
	0.75	0.95441	0.67144	−0.63935
	0.90	0.93515	0.61661	−0.52184

There are two built-in values for the `ctitle` key: `ctitle=*`, which forms the title from the function being tabulated, and `ctitle=**` which uses the function and vv-list for the title. Obviously these, particularly the latter, could easily become too long to be useful. An example of `ctitle=**` is presented later in

§2.4.5, but inclusion of the vv-list in the title in the next example would be pointless since the row and column variables of the table are the only members of the vv-list; `ctitle=*` suffices:

```
\tabulate[rspec={n,1,5},rround=0,
          cspec={m,1,4},chstyle=2,ctitle=*]
  { \cos(m\pi/n) }[n=3,m=2][*4]
```

⟹

	$\cos(m\pi/n)$			
n	$m=2$	$m=3$	$m=4$	$m=5$
3	-0.5000	-1.0000	-0.5000	0.5000
4	0.0000	-0.7071	-1.0000	-0.7071
5	0.3090	-0.3090	-0.8090	-1.0000
6	0.5000	0.0000	-0.5000	-0.8660
7	0.6235	0.2225	-0.2225	-0.6235

2.4.2 Between header & title: `csubttl`

Some tables need more header or title material than can be comfortably accommodated in either row alone. For examples, see *HMF* Tables 7.9, 17.7, 21.1, and 26.7. One way of handling this problem is to resort to more complicated environments in header and title rows. Another, more direct way, is to insert a row between the header and title rows by means of the key `csubttl`, a contraction of 'c(olumn variable) subtitle'. (In version 2 of `numerica-tables` the name `cmidrow` was used; that is still available, but deprecated.) Like `chead` and `ctitle`, `csubttl` is limited to the span of the column variable (or function-value) columns only. The content of `csubttl` is entirely up to the user, including insertion of sufficient `&` characters and math delimiters (if required).

An example where the subtitle row serves an explanatory role is shown in the next table. I have used `\sfrac` from the `xfrac` package for neater fractions:

```
\tabulate[ff,rspec={x,1,6,0},chnudge={22.5,18},
 ctitle=\text{Hyperbolic functions},
 csubttl=\multicolumn{3}{c}{\small$\sinh x=\sfrac12
   (e^x-e^{-x}),\ \cosh x=\sfrac12(e^x+e^{-x})$}]
  { \sfrac12\,e^x, \sinh x, \cosh x } [{x}=0][8*]
```

⟹

	Hyperbolic functions		
	$\sinh x = {}^1\!/_2(e^x - e^{-x}),\ \cosh x = {}^1\!/_2(e^x + e^{-x})$		
x	${}^1\!/_2\, e^x$	$\sinh x$	$\cosh x$
0	0.50000000	0.00000000	1.00000000
1	1.35914091	1.17520119	1.54308063
2	3.69452805	3.62686041	3.76219569
3	10.04276846	10.01787493	10.06766200
4	27.29907502	27.28991720	27.30823284
5	74.20657955	74.20321058	74.20994852

2.4.3 Suppress/show header row

By default the header row in a table is shown. It carries essential information as to the table's contents, but there are occasions when it should be suppressed. An example occurs in §2.1.2.2 where a table listing fractions of π and their values is shown. The header there serves no purpose. To suppress it, use the `headless`[1] setting.

```
\def\mydata{\sfrac14\,\pi,\sfrac13\,\pi,\sfrac12\,\pi,
            \sfrac23\,\pi,\sfrac34\,\pi,\pi}
\tabulate[rdata=\mydata,rverb=1,rvar=k,headless,
          norules]{ k }[*]
```

⟹

$^{1}/_{4}\,\pi$	0.785398
$^{1}/_{3}\,\pi$	1.047198
$^{1}/_{2}\,\pi$	1.570796
$^{2}/_{3}\,\pi$	2.094395
$^{3}/_{4}\,\pi$	2.356194
π	3.141593

2.4.4 Footer row: `foot`

Some tables have a footer row and `numerica-tables` allows such a row to be inserted, but its entire content, with two exceptions, is the responsibility of the user, including insertion of the necessary number of tab characters `&`. This will usually be 1 less than the total number of columns (including row variable columns) in the table – or some adjustment thereof if you use `\multicolumn`. (*HMF* uses the footer mainly for cryptic descriptions of the accuracy and needs of interpolation methods.)

You can put into the footer what you wish with the setting `foot=<tokens>`. There are two exceptions:

- when `foot=''` (two single quote marks), suggesting ditto marks, the footer row is set equal to the header row with one change, the backslash in row variable headers of the form $n\backslash m$ are converted to forward slashes, e.g. n/m;
- when `foot=*`, the footer is set equal to the header in *reversed* order – the last item first, and so on. This is useful for tabulating complementary functions like sine and cosine or, more generally, $f(x)$ and $g(x)$ where $g(x) = f(k - x)$ for some constant k. Values for the complementary function are read from the bottom up and require a reversed row variable column on the right of the table; see the third example in §2.4.6.

---

[1] There was a mix-up in the documentation of version 3.1 of `numerica-tables` between what was entered in the table of whole-of-table settings and what was described in this section.

Footer functions In versions of `numerica-tables` before version 3 it was possible to perform certain simple operations on columns – calculate the sum, the average and maximum and minimum values. This is no longer so in version 3. Not only does it seem tangential to the primary function of the `\tabulate` command but it was also acutely dependent on the format of the numbers being operated on – a change in the number-format option could cause a LaTeX error.

2.4.5 Horizontal rules: `rules`

The `booktabs` package which `numerica-tables` uses is most emphatic that one should '1. Never, ever use vertical rules. 2. Never use double rules.' Most of the tables proper in *HMF* lack rules of any kind although closer inspection shows smaller tables within the text generally *are* delimited by horizontal rules (often also with vertical rules). In the various examples in the present document I have used horizontal rules because these too are tables within text. Some form of delineation seems necessary although I think I have 'over ruled'. The question should always be: is a rule necessary at all? Usually, less is more. (Although many of *HMF*'s tables are inelegantly typeset, I have used it as a valuable resource for the variety of structures needed to present a multitude of different kinds of numerical data.)

The `rules` key allows one to specify precisely which rules are used. The content of the key is a 'word' – a sequence of letters – where the characters have the significance and default thicknesses – from the `booktabs` package – shown in Table 2.7. (To adjust the thickness of rules used, see §2.4.5.3 below.) No particular order is required in the `rules` specification; `rules=hBT` would work as well as `rules=ThB` but the latter, reflecting the order of the rules in the table, is simpler to grasp at a glance.

In previous versions of `numerica-tables` the initial value of this key was `rules=ThB` so that if the `rules` key was not explicitly set then a rule was automatically drawn at the top of the table, at the bottom of the table, and beneath the header row. From version 3.2, no initial value is assigned to `rules`. Instead, a package option of the same name can be set (see §1) which is initialized to `rules=ThB` to maintain previous behaviour. What is gained is that by assigning a different value to the package option it is now possible for a user to alter the default behaviour. A `norules` package option is also available if you want your tables free of rules by default. The default set by the package option can always be overridden for an individual table by means of the `rules` *setting*.

2.4.5.1 Header and footer rules

The header (resp. footer) rule specified by including `h` (resp. `f`) in the `rules` specification spans the whole table, from the left of the first column to the right of the last column. It can give a better visual appearance sometimes to trim (shorten) these rules both left and right. This is specified by priming the `h` (resp. `f`) in the specification. Thus `rules=hf` will produce header and footer rules spanning the whole table; `rules=h'f'` will produce slightly shorter header

Table 2.7: Rules. (In the 'span' column, 'fv'=function-value; 'rv'=row variable.)

char	rule	position	span	trim	default rule thickness
`T`	top	above table	table		`\heavyrulewidth=.08em`
`t`	title	below title	fv cols	`.5em`	`\cmidrulewidth=.03em`
`s`	subtitle	below subtitle	fv cols (1 rv col.)	`.5em`	`\cmidrulewidth=.03em`
			table (2 rv cols)	`.5em`	`\lightrulewidth=.05em`
`h`	header	below header	table		`\lightrulewidth=.05em`
`h'`				`.5em`	`\cmidrulewidth=.03em`
`f`	footer	above footer	table		`\lightrulewidth=.05em`
`f'`				`.5em`	`\cmidrulewidth=.03em`
`B`	bottom	below table	table		`\heavyrulewidth=.08em`

and footer rules trimmed by the default 0.5 em at both ends – unless some other trim value has been specified; see §2.4.5.3 below.

2.4.5.2 Title and subtitle rules

If `t` is included in the `rules` specification, then a rule will be drawn beneath the title. This will span the function-value columns only and be trimmed at each end (by default by 0.5 em). If you are also using a subtitle row, between title and header rows, and want a rule beneath that too, then include `s` in the setting – for instance `rules=TtshB`. (For legacy reasons, `m` – from 'midrow' – can also be used instead of `s`.) To my eye rules beneath *both* title and subtitle don't work; a rule beneath the subtitle alone gives a better result – if a rule is needed at all.

The span of the subtitle rule depends on whether the row variable column is placed only on one side of the table or on both (`rpos<2` or `rpos>2`). If on only one then the rule spans only the function-value columns – like the title rule. If on both then the subtitle rule spans the table but is trimmed by 0.5 em at each end. To the author's eye, the extra span in this second case gives a better sense of the row variable columns as part of the table as a whole rather than as optional 'clip ons' at the edges.

In the example table below, a rule for the column title has been specified (the `t` in the setting `rules=TthB`). Also note the use of `ctitle=**`. The formula contains an extra parameter a, assigned a value in the vv-list, so that it now makes sense to display the vv-list in the column title (the braces around `k` and `x` in the vv-list ensure they don't display).

```
\tabulate[rspec={x,0.25,5,2},cspec={k,0.25,3,2},
          rhnudge=9,chstyle=2,ctitle=**,rules=TthB]
  { a\sin kx }[a=2/\pi,{k}=3,{x}=0.25][*]
```

$\Longrightarrow$

	$a\sin kx, \quad (a=2/\pi)$		
x	$k=3.00$	$k=3.25$	$k=3.50$
0.25	0.433945	0.462191	0.488633
0.50	0.635025	0.635685	0.626425
0.75	0.495337	0.412111	0.314439
1.00	0.089840	−0.068879	−0.223316
1.25	−0.363867	−0.506845	−0.600729

2.4.5.3 Trim and thickness of rules

The trim applied to rules – the amount cut from the ends – is the `booktabs`' default value, 0.5 em. To change it, enter the command

```
\setlength\cmidrulekern{<trim>}
```

in the preamble where `<trim>` is a length, something like `0.5em`, `2.5mm`, etc. (Or, one could write, e.g. `\cmidrulekern=2.5mm`.) The new default will apply to the settings `h'`, `f'`, `t` and sometimes `s`, since these all use `booktabs`' `\cmidrule`.

To change the thickness of a rule from its default value, enter new values in the preamble for any or all of `\heavyrulewidth`, `\lightrulewidth`, `\cmidrulewidth`, e.g.

```
\setlength\heavyrulewidth{<width>}
```

where `<width>` is a length. The values listed in Table 2.7 are the default values, from the `booktabs` package, used in `numerica-tables`.

2.4.6 Second row variable column: `rpos=3,4`

The settings `rpos=0,1,2` have been discussed earlier; `rpos=3` duplicates the row variable column on the right of the table, with the proviso that if the header of the row variable column on the left is automatically generated and like $x\backslash k$, the header on the right is changed to a form like k/x. Repeating the first example of §2.2.2.2 but with the setting `rpos=3`,

```
\tabulate[rpos=3,rspec={x,0.2,5},
  ccspec={k,2,cstop=7,chnudge=18]
    { \sin kx }[k=3,x=0.2][4*]
```

$\Longrightarrow$

$x\backslash k$	3	5	7	k/x
0.2	0.5646	0.8415	0.9854	0.2
0.4	0.9320	0.9093	0.3350	0.4
0.6	0.9738	0.1411	−0.8716	0.6
0.8	0.6755	−0.7568	−0.6313	0.8
1.0	0.1411	−0.9589	0.6570	1.0

An example of using `rpos=3` in a multi-function table is provided near the end of §2.3.

The `rpos` key can also take the value 4. `rpos=4` adds the row variable column to both left and right sides of the table, but the values displayed in the right column are a *function of those in the left column* (`rpos=3` corresponds to the function being the identity). The value of the key `rvar'` specifies the function used.

```
\tabulate[rpos=4,rspec={x,0.5,5,2},rhnudge=9,
            cspec={k,2,3},chstyle=2,ctitle=*,
              rvar'=x^2,rhnudge'=4]
    {  \sin(kx^2) }[k=3,x=1][4*]
```

$\Longrightarrow$

		$\sin(kx^2)$		
x	$k=3$	$k=5$	$k=7$	x^2
1.00	0.1411	−0.9589	0.6570	1.00
1.50	0.4500	−0.9678	−0.0420	2.25
2.00	−0.5366	0.9129	0.2709	4.00
2.50	−0.0994	−0.1652	−0.2302	6.25
3.00	0.9564	0.8509	0.1674	9.00

If no value is given to `rvar'` (or if `rvar'=*`) then the row variable column on the right of the table is the left column reversed (with the possible exception of its header).

- Note: `rpos=4` is incompatible with the `transpose` setting. If both settings are used, the table will be transposed as if `rpos=3`.

When `chstyle` is not zero, as in the example above, and `rhead'` is omitted or empty, the header for the right-hand row variable column is automatically generated, displaying the value of the `rvar'` key. The positioning of the right-hand row variable header can be adjusted with the use of `rhnudge'` as in the example. If `chstyle=0`, the header is up to the user to supply by assigning a value to the key `rhead'`. In all cases, `rhead'` will override any automatically generated header.

The sine and cosine are complementary functions; when working in degrees (the `o` setting in the next example), $\cos\theta = \sin(90-\theta)$. We can exploit this fact to halve the table size needed to tabulate the two functions. The following 'toy' example (see *HMF* Tables 4.10–4.12 for the real thing) of a multi-function table (hence the `ff` setting) also gives an example of the use of the `foot=*` setting, and the primed versions of header and footer rules. Note also the use of an expression in the third element of `rspec`. The `rhead'` setting is of no significance; it simply avoids a large block of white space.

```
\tabulate[ff,o,rpos=4,rules=Th'f'B,foot=*,calign=c,
           rspec={\theta,9,1+45/9,0},
             rvar'=90-\theta,rhead'=-]
  { \sin\theta,\cos\theta }[\theta=0][*]
```

$\Longrightarrow$

θ	$\sin\theta$	$\cos\theta$	–
0	0.000000	1.000000	90
9	0.156434	0.987688	81
18	0.309017	0.951057	72
27	0.453990	0.891007	63
36	0.587785	0.809017	54
45	0.707107	0.707107	45
–	$\cos\theta$	$\sin\theta$	θ

The values of sines from 0 to 45 degrees are read downwards from the first column of function-values, and from 45 to 90 degrees are read upwards from the second column of function-values. For cosines it is downwards from the second column and upwards from the first column. The reversed footer line indicates the change of columns to use.

Although there is a significant space saving with tables like this, they are not 'kind to the reader'. They require a certain concentration to read and should be avoided unless space is seriously constrained. *HMF* Tables 6.1 and 6.2 are tables of the gamma function and its relatives where $y = x - 1$ is used in the row variable column on the right (stemming from $x! = \Gamma(x - 1)$); *HMF* Table 6.5 in effect uses $\langle 1/x \rangle$ (the nearest integer to $1/x$) for the row variable on the right.

2.4.7 Separating blocks of rows: `rbloc`

Readability of long columns of figures can be aided by adding extra white space between blocks of rows. This is achieved with the `rbloc` key:

```
rbloc = <comma list of positive integers>
```

specifies how many rows belong to each block. For example, `rbloc={5,5,6}` breaks the table into blocks of 5 rows, 5 rows, then 6 rows. If the number of rows in the table is greater than the sum of the entries in the comma list, then division into blocks continues as specified by the last entry in the comma list. Thus `rbloc=5` (strictly `rbloc={5}` but the braces can be omitted in this case since no comma is enclosed) divides a table into blocks of 5 rows; `rbloc={1,5}` divides a table into 1 row followed by blocks of 5 rows. A division of this kind may be appropriate when, say, the row variable runs from 0 to 1 in increments of 0.1 – there are 11 rows of which the first (when the row variable is zero) may have distinctive values.

The dominant practice in *HMF* is division into blocks of (generally) 5 rows, many of which start with a zero value for the row variable. Rather than isolate this initial value, they include it in the first block of 5, then continue with blocks of 5 until a single isolated row is left at the bottom of the page or the table. There seems to be a psychological need to finish a page or table with the row variable set to a nice round number. Thus: tabulate from 0 to 10 rather than 0 to 9, from 0 to 1 rather than 0 to 0.9, and even from 0 to 30 or 0 to 2 rather than 0 to 29 or 0 to 1.9. Using blocks of 5 the consequence is that there is always an isolated line at the end – a kind of punctuation mark to signal the end of the page or the table.

In the next example I have divided the rows into blocks of 4 but separated the special values of the first and last rows by setting `rbloc={1,4}`.

```
\tabulate[ff,o,rspec={\theta,10,1+90/10,0},rbloc={1,4}]
    { \sin\theta, \cos\theta }[\theta=0][*]
```

	θ	$\sin\theta$	$\cos\theta$
	0	0.000000	1.000000
	10	0.173648	0.984808
	20	0.342020	0.939693
	30	0.500000	0.866025
$\Longrightarrow$	40	0.642788	0.766044
	50	0.766044	0.642788
	60	0.866025	0.500000
	70	0.939693	0.342020
	80	0.984808	0.173648
	90	1.000000	0.000000

2.4.7.1 Adjusting the extra space `rblocsep`

By default `numerica-tables` sets the extra space between blocks of rows at `1 ex`. This value can easily by changed with the setting `rblocsep=<length>`. The units need to be included in the specification.

2.4.8 'Horizontal' tables: `transpose`

Traditionally in a book of mathematical tables the function values are arranged vertically in columns. This is why the row variable has been given primacy in `numerica-tables`. But that may not always be what is wanted nor make the best use of available space. From version 3.2.0 of `numerica-tables` there is a setting, `transpose`, that will convert a 'vertical' table (function values in columns) to a 'horizontal' table (function values in rows). The table is *constructed* as if the function values will be listed vertically in columns, but with `transpose` entered in the settings option the table is displayed with the function values listed in rows.

(Strictly, `transpose` 'is really' `transpose=1`, but `transpose` alone suffices; `transpose=0` corresponds to the default situation of function values in columns.)

With transposition, the row variable column and header row are interchanged. The building of the row variable column and header row occurs *before* transposition. Hence `rround` and `chround` are applied to these elements *before* transposition. It is the rounded values of the row and column variables that are used to calculate function values. But 'cosmetic effects' like placement of the (new) row variable column or (new) header row, as well as header styling (`chstyle`), alignment (`ralign`, `calign`), nudging (`rhnudge`, `chnudge`), fonts (`rfont`, `chfont`), and choice of display-, text- or scriptstyle (`rmath`, `chmath`) are applied *after* transposition. A footer row is created *after* transposition.

- Transposition does not work for tables using a second row variable column with `rpos=4`. Such a table *is* transposed, but treated as if `rpos=3`.

In a 'shallow' transposed table – like one with a single row of function values – rules should be used sparingly if at all. See the example at §2.1 where the `norules` setting is used. In the following example, with three function-value rows resulting from the transposition, a single header rule suffices.

```
\tabulate[rspec={x,0.2,5},ralign=c,cspec={k,0.25,3,2},
           chstyle=1,rules=h',transpose]
  { \sin kx }[k=3,x=0.2][*]
```

$\Longrightarrow$

k	$x = 0.2$	0.4	0.6	0.8	1.0
3.00	0.564642	0.932039	0.973848	0.675463	0.141120
3.25	0.605186	0.963558	0.928960	0.515501	−0.108195
3.50	0.644218	0.985450	0.863209	0.334988	−0.350783

2.4.9 Table placement

Tables can be nudged vertically with the LaTeX commands `\bigskip`, `\medskip`, `\smallskip`, usually about 1, 1/2 and 1/4 line spaces (with stretch and shrink). `booktabs` provides `\abovetopsep` and `\belowbottomsep`, both set by default to `0ex` and easily changed by writing, e.g. `\setlength\abovetopsep{1.25ex}` (or `\abovetopsep=1.25ex`) if you want to insert `1.25ex` of space above the table (perhaps to fit captions).

2.4.9.1 Vertical alignment

By writing `valign=<char>` where `<char>` is one of `t`, `m` or `b` the vertical alignment of the table can be set relative to the text baseline: `t` aligns the top of the table, `b` the bottom of the table, and `m` the middle of the table with the text baseline. By default `valign=m` is set. Repeating an example from earlier (§2.1) I have added letters A, B, C to show where the baseline is. Clearly the top, middle and bottom of the respective tables aligns with the baseline.

```
A \tabulate[valign=t,rvar=x,rstep=0.2,rows=5]
  { \sin x/\cos x }[x=0][*] \quad
B \tabulate[rspec={x,0.2,1/0.2}]
  { \tan x }[x=0][*] \quad
C \tabulate[valign=b,rspec={x,0.2,5}]
  { \sqrt{\sec^2 x - 1} }[x=0][*]
```

$\Longrightarrow$ A

x	$\sin x/\cos x$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408

B

x	$\tan x$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408

C

x	$\sqrt{\sec^2 x - 1}$
0.2	0.202710
0.4	0.422793
0.6	0.684137
0.8	1.029639
1.0	1.557408

As explained in §2.1.3.1, tables can be adjoined to give the appearance of a single larger table. If tables with different numbers of rows are adjoined in this manner, then a middle alignment fails and a top alignment is necessary (so that the header rows of the tables align).

2.5 Formatting function values

In previous tables in this document, function values have generally been limited to a fairly narrow range of values. What happens when they span orders of magnitude? Can scientific notation, expressly designed to cope with such differing orders of magnitude, be accommodated in a table in a natural way? Can rows or columns – or individual cells – be rounded to different rounding values? Can we indicate differences? Or form tables of function values in fraction form? These and similar questions are our concern here.

2.5.1 Trailing optional argument

The primary tool for function-value formatting is the trailing optional argument of the `\tabulate` command where the rounding value is specified, padding with zeros is set or not (generally *set* in tables), scientific notation is set or not, and fraction-form output can be specified.

2.5.1.1 Fraction-form output

Function values in a table can be presented in fraction form, but such output requires far more computation than other forms since finding denominators at the specified accuracy is an iterative process that needs doing for every function value. But for small tables it is feasible. In the tables below, approximations to small positive and inverse powers of π are listed to 2 and 4 decimal places of accuracy. All the powers listed can be approximated to 4-place accuracy by 3-figure denominators (and π^2 by a 2-figure denominator).

Table 2.8: Formatting function values

key	type	meaning	initial
`(pad)`	int	t-notation phantom padding	
`signs`	int	sign handling for function values	`0`
`diffs`	int	insert differences, pre-pad with 0s	`0`
`round`	tokens	row/col. dependent rounding value	
`Q?`	tokens	special cell conditional	
`A!`	tokens	special cell formatting	

```
\def\mypi{\pi,\pi^2,\pi^3,\pi^{\sfrac12},\pi^{\sfrac13}}
\tabulate[rdata=\mypi,rverb=1,rpos=1,rvar=k,ralign=l,
          chead={\small $2$ places}]{ k }[2/s] \qquad
\tabulate[rdata=\mypi,rverb=1,rpos=1,rvar=k,ralign=l,
          chead={\small $4$ places}]{ k }[4/s]
```

$\Longrightarrow$

‘	2 places
π	$^{22}/_{7}$
π^2	$^{148}/_{15}$
π^3	$^{2760}/_{89}$
$\pi^{1/2}$	$^{23}/_{13}$
$\pi^{1/3}$	$^{19}/_{13}$

‘	4 places
π	$^{355}/_{113}$
π^2	$^{227}/_{23}$
π^3	$^{4930}/_{159}$
$\pi^{1/2}$	$^{257}/_{145}$
$\pi^{1/3}$	$^{186}/_{127}$

A second example shows that all four of `numerica`'s built-in constants and their first few inverse powers can be approximated to 5 decimal places with 3-figure denominators:

```
\tabulate[rdata={\pi,e,\phi,\gamma},rverb=1,/max=1000,
    rvar=k,cspec={n,1,4},chstyle=3,chnudge=9,rules=TthB,
    ctitle=\abs{k^{\sfrac1n}-p/q}<0.5\times10^{-5}]
  { k^{\sfrac1n} }[n=1,k=1][/s5]
```

$\Longrightarrow$

	$\lvert k^{1/n}-p/q\rvert < 0.5\times10^{-5}$			
k	$k^{1/1}$	$k^{1/2}$	$k^{1/3}$	$k^{1/4}$
π	$^{355}/_{113}$	$^{296}/_{167}$	$^{517}/_{353}$	$^{667}/_{501}$
e	$^{1264}/_{465}$	$^{582}/_{353}$	$^{1210}/_{867}$	$^{217}/_{169}$
ϕ	$^{610}/_{377}$	$^{491}/_{386}$	$^{668}/_{569}$	$^{397}/_{352}$
γ	$^{228}/_{395}$	$^{487}/_{641}$	$^{194}/_{233}$	$^{421}/_{483}$

2.5.1.2 Scientific notation

Scientific notation – put `x` in the trailing optional argument – is generally inappropriate for use in tables; see the first table below. Entering `xx` (second table) so that the notation extends to numbers in the range $[1, 10)$ helps, particularly with the *left* alignment of the function values, but the result is wasteful of space and the repetition of the '$\times 10$' is distracting and would be more so for a larger table. The `x` specification should be used in tables, if at all, only for *small* tables and special cases. The `t` option is much preferred; see §2.5.2 following.

```
\tabulate[rspec={x,1,6,0},chnudge=58]
  { e^x}[x=-2][*x]\qquad
\tabulate[rspec={x,1,6,0},calign=l,chnudge=45]
  { e^x}[x=-2][*xx]
```

$\Longrightarrow$

x	e^x
-2	1.353353×10^{-1}
-1	3.678794×10^{-1}
0	1.000000
1	2.718282
2	7.389056
3	2.008554×10^{1}

x	e^x
-2	1.353353×10^{-1}
-1	3.678794×10^{-1}
0	1.000000×10^{0}
1	2.718282×10^{0}
2	7.389056×10^{0}
3	2.008554×10^{1}

2.5.2 The `t` option

HMF uses a special notation for coping with function values spanning different orders of magnitude. This notation can be invoked by inserting `t` in the trailing optional argument. For the previous two tables the notation gives a more compact and visually appealing result:

```
\tabulate[rspec={x,1,2*3+1,0},chnudge=24]
  { e^x}[x=-3][*t]\qquad
\tabulate[rspec={x,1,2*3+1,0},chnudge=24]
  { e^x}[x=-3][*tt]
```

$\Longrightarrow$

x	e^x
-3	$(-2)\,4.978707$
-2	$(-1)\,1.353353$
-1	$(-1)\,3.678794$
0	1.000000
1	2.718282
2	7.389056
3	$(1)\,2.008554$

x	e^x
-3	$(-2)\,4.978707$
-2	$(-1)\,1.353353$
-1	$(-1)\,3.678794$
0	$(0)\,1.000000$
1	$(0)\,2.718282$
2	$(0)\,7.389056$
3	$(1)\,2.008554$

Doubling the `t` in the trailing optional argument (second table) extends the notation to numbers in the range $[1, 10)$ – as doubling `x` extends normal scientific notation to this range.

2.5.2.1 Padding the exponent: `(pad)`

In the second table one might quibble at the lack of alignment of the left parentheses. *HMF* tends to align these and `numerica-tables` offers the setting

```
(pad) = <integer>
```

to achieve the effect. (The parentheses are part of the key – a reminder of the `t`-form of scientific notation.) `<integer>` is the number of digits/characters to pad to. Repeating the last two tables with the setting `(pad)=2` produces the following results:

```
\tabulate[rspec={x,1,2*3+1,0},chnudge=24,(pad)=2]
  { e^x}[x=-3][*t]\qquad
\tabulate[rspec={x,1,2*3+1,0},chnudge=24,(pad)=2]
  { e^x}[x=-3][*tt]
```

$\Longrightarrow$

x	e^x
-3	$(-2)\,4.978707$
-2	$(-1)\,1.353353$
-1	$(-1)\,3.678794$
0	1.000000
1	2.718282
2	7.389056
3	$(\ \ 1)\,2.008554$

x	e^x
-3	$(-2)\,4.978707$
-2	$(-1)\,1.353353$
-1	$(-1)\,3.678794$
0	$(\ \ 0)\,1.000000$
1	$(\ \ 0)\,2.718282$
2	$(\ \ 0)\,7.389056$
3	$(\ \ 1)\,2.008554$

Examples in *HMF* of the style exemplified by the first table are, among others, Tables 8.6, 9.2, 20.1, and of the style exemplified by the second table, among many, Tables 9.9, 10.5, 13.1, 14.1, 19.1. But note:

- the `(pad)` setting is relevant only when the `t` option is used in the trailing number-format argument of the `\tabulate` command and should have no effect otherwise.

2.5.2.2 Accommodating signs in the `t`-notation

Instead of e^x as the test function, use $e^x - 1$. Now there are positive, zero and negative function values to contend with. Recall that in the `t`-notation the *exponent* is the parenthesized integer part of a number and the *significand* the following decimal figures. `numerica-tables` offers the `signs` key to align (or not) the exponents. The setting is

```
signs = <integer>
```

Besides the do-nothing default (`signs=0` or omitted), there are four effective values for `<integer>`:

- `signs=2` inserts a $+$ sign between exponent and significand of every non-negative number;
- `signs=1` inserts a $+$ sign between exponent and significand of every non-negative number that immediately precedes or follows a negative number;

- `signs=-1` inserts a + sign between exponent and significand of any non-negative number that immediately precedes or follows a negative number, and inserts a *phantom* + sign between exponent and significand of every other non-negative number;
- `signs=-2` inserts a *phantom* + sign between exponent and significand of every non-negative number;

In the following examples, `signs=-2`, `signs=-1` and `signs=2`, all give acceptable results.

```
\tabulate[rspec={x,1,2*3+1,0},(pad)=2,signs=-2]
  { e^x-1}[x=-3][4*tt] \qquad
\tabulate[rspec={x,1,2*3+1,0},(pad)=2,signs=-1]
  { e^x-1}[x=-3][4*tt] \qquad
\tabulate[rspec={x,1,2*3+1,0},(pad)=2,signs=2]
  { e^x-1}[x=-3][4*tt]
```

⟹

x	$e^x - 1$
−3	(−1) −9.5021
−2	(−1) −8.6466
−1	(−1) −6.3212
0	(0) 0.0000
1	(0) 1.7183
2	(0) 6.3891
3	(1) 1.9086

x	$e^x - 1$
−3	(−1) −9.5021
−2	(−1) −8.6466
−1	(−1) −6.3212
0	(0) +0.0000
1	(0) 1.7183
2	(0) 6.3891
3	(1) 1.9086

x	$e^x - 1$
−3	(−1) −9.5021
−2	(−1) −8.6466
−1	(−1) −6.3212
0	(0) +0.0000
1	(0) +1.7183
2	(0) +6.3891
3	(1) +1.9086

In *HMF* Table 23.2 illustrates `signs=-2`; Tables 10.1, 13.1, 14.1, 19.1 among many others illustrate `signs=-1`; and Tables 9.4, 10.6, 20.2, 22.11 among others illustrate `signs=2`.

For `signs=1`, see the second table in the next example.

2.5.3 Indicating signs outside the t-notation

The `signs` key is not limited to the `t`-notation. In the following tables where the notation is not used, positive values for the key, including `signs=1`, give good results. The third table, corresponding to `signs=0`, shows that the comment extends to *non-negative* values of the `signs` key.

```
\tabulate[rspec={x,0.1,7},(pad)=2,signs=2]
  { 10\sin 5x}[x=-0.3][*4]\qquad
\tabulate[rspec={x,0.1,7},(pad)=2,signs=1]
  { 10\sin 5x}[x=-0.3][*4]\qquad
\tabulate[rspec={x,0.1,7},(pad)=2]
  { 10\sin 5x}[x=-0.3][*4]
```

⟹

x	$10\sin 5x$
-0.3	-9.9749
-0.2	-8.4147
-0.1	-4.7943
0.0	$+0.0000$
0.1	$+4.7943$
0.2	$+8.4147$
0.3	$+9.9749$

x	$10\sin 5x$
-0.3	-9.9749
-0.2	-8.4147
-0.1	-4.7943
0.0	$+0.0000$
0.1	4.7943
0.2	8.4147
0.3	9.9749

x	$10\sin 5x$
-0.3	-9.9749
-0.2	-8.4147
-0.1	-4.7943
0.0	0.0000
0.1	4.7943
0.2	8.4147
0.3	9.9749

HMF seems to use `signs=2` when the sign of the function values changes every few entries and `signs=1` when there are runs of entries of the same sign. Over the range tabulated here for $10\sin 5x$, they would use the middle table of the three, `signs=1`.

2.5.4 Cell-, row-, column-dependent rounding

In §2.5.1.1, we created two tables of fraction-form approximations to simple powers of π, one accurate to two places of decimals, one to four. From version 3.1 (as distinct from version 3.0) `numerica-tables` offers the means of producing tables with rounding values depending on position in the table. This is effected through the key `round` which sets the rounding value as a function of row and column variables. In practice this usually means dependence on row or column variable alone rather than both. In the example below, the rounding equals the row variable value, `round=r`, producing fractional approximations to simple powers of π at rounding values from 1 to 5, and learn that all these powers can be approximated to 5 decimal places with 3 figure denominators – π^2 only just. (The command `\abs` used in `ctitle` is defined in `numerica`.)

```
\tabulate[ff,rspec={r,1,5},round=r,/max=999,chstyle=2,
      ctitle=\abs{\pi^k-\sfrac mn}<0.5\times10^{-r}]
   { \pi,\pi^2,\pi^3,\pi^{1/2},\pi^{1/3}}[r=1][/s]
```

⟹

	$\vert\pi^k - {}^m/_n\vert < 0.5\times 10^{-r}$				
r	π	π^2	π^3	$\pi^{1/2}$	$\pi^{1/3}$
1	$^{19}/_{6}$	$^{59}/_{6}$	$^{31}/_{1}$	$^{7}/_{4}$	$^{3}/_{2}$
2	$^{22}/_{7}$	$^{148}/_{15}$	$^{2760}/_{89}$	$^{23}/_{13}$	$^{19}/_{13}$
3	$^{267}/_{85}$	$^{227}/_{23}$	$^{4589}/_{148}$	$^{39}/_{22}$	$^{41}/_{28}$
4	$^{355}/_{113}$	$^{227}/_{23}$	$^{4930}/_{159}$	$^{257}/_{145}$	$^{186}/_{127}$
5	$^{355}/_{113}$	$^{9840}/_{997}$	$^{14821}/_{478}$	$^{296}/_{167}$	$^{517}/_{353}$

Another place where a variable rounding value can be of value is when a function being tabulated changes slowly for each step in the row variable value. The value of the cosine for instance changes from 1.0000 to 0.9848 between 0° and 10°. *Part* of a table of the cosine might be something like the following, where values in the initial rows of the table are rounded to a higher value than in later rows. In the example `round` is set to an expression in the row variable `\theta` involving the boolean expressions `\theta<11` and `\theta>10` which evaluate to 0 or 1 depending on the value of `\theta`. Thus `round` takes the value 6 for the initial rows of the table and the value 4 thereafter.

```
\tabulate[o,rspec={\theta,1,6},calign=l,chnudge=15,
          round=6(\theta<11)+4(\theta>10)]
  { \cos\theta }[\theta=8]
```

	θ	$\cos\theta$
	8	0.990268
	9	0.987688
⟹	10	0.984808
	11	0.9816
	12	0.9781
	13	0.9744

2.5.5 Differences: `diffs`

In fine-grained tables where function values change only slowly from entry to entry it can be helpful to include a difference entry between function-value entries as an aid to interpolation (and a test of eyesight). By entering

```
diffs = <non-negative integer>
```

the `\tabulate` command will include differences in a table. The `<non-negative integer>` is the maximum number of digits in a difference.

```
\tabulate[rspec={x,0.01,6,2},diffs=3,
            rhnudge=9,chnudge=21]
    { \sinh x }[x=1,k=1][*4] \qquad
\tabulate[rspec={x,0.01,6,2},diffs=2,
            rhnudge=9,chnudge=25]
    { \sinh x }[x=1][*4]\qquad
\tabulate[rspec={x,0.01,6,2},diffs=4,
            rhnudge=9,chnudge=30]
    { \sinh x }[x=1][*4]
```

	x	$\sinh x$
	1.00	1.1752
	1.01	1.1907 ^{155}
⟹	1.02	1.2063 ^{156}
	1.03	1.2220 ^{157}
	1.04	1.2379 ^{159}
	1.05	1.2539 ^{160}

x	$\sinh x$
1.00	1.1752
1.01	1.1907 ^{155}
1.02	1.2063 ^{156}
1.03	1.2220 ^{157}
1.04	1.2379 ^{159}
1.05	1.2539 ^{160}

x	$\sinh x$
1.00	1.1752
1.01	1.1907 ^{0155}
1.02	1.2063 ^{0156}
1.03	1.2220 ^{0157}
1.04	1.2379 ^{0159}
1.05	1.2539 ^{0160}

I have deliberately chosen the settings in the first table – `diffs=3` – to give a good result. With the default right alignment of the function-value columns, it is easy to get this wrong. The evidence will be either in the misalignment of the first row of function values (second table) or unnecessary padding of differences with leading zeros (third table). It is a good idea to create your table first, see how function values change between successive rows and judge how many digits there will be in a difference. When the `diffs` setting is too small, function values in the first row are misaligned, the amount depending on how much too small. (A left alignment of the function-value column is another way of tackling this issue.) When the `diffs` setting is too big, alignment is fine but differences are padded with unnecessary leading zeros, meaning the column header will need a bigger nudge to bring *it* into alignment.

In the next example the function is *decreasing*, showing how it is the *absolute value of the difference* between successive function values that is tabulated. A difference is always a non-negative value.

```
\tabulate[rspec={x,0.01,6,2},diffs=2,
            rhnudge=9,chnudge=21]
  { e^{-x^2} }[x=0.05][*4]
```

	x	e^{-x^2}	
	0.05	0.9975	
	0.06	0.9964	11
$\Longrightarrow$	0.07	0.9951	13
	0.08	0.9936	15
	0.09	0.9919	17
	0.10	0.9900	19

2.5.6 Formatting special values: `Q?` and `A!`

You may wish to highlight or display in some special way a particular function value or values. `\nmcTabulate` has two related settings that enable this: `Q?=<tokens>` and `A!=<tokens>`. As the names suggest: Question? and Answer!

The question should be an expression that `l3fp` can digest and produce a boolean answer to (1 for 'true', 0 for 'false'). *This is not a LaTeX expression but an* `l3fp` *expression.*[2] For the user it should suffice to know that an expression formed from decimal numbers (but `l3fp` knows only the decimal dot), parentheses `( )`, the familiar arithmetic symbols, `+`, `-`, `*`, `/` and `^`, relation symbols `<`, `>`, `=` and combinations like `!=` (for $\neq$), `>=` (for $\geq$), and `<=` (for $\leq$) will be digested by `l3fp`. In addition there are `||` for logical Or, `&&` for logical And, and `!` for logical Not; `exp(1)` for e and `pi` (no backslash) for π. `numerica-tables` provides `MAX` and `MIN` for the maximum and minimum function values tabulated, and uses `@` to denote the function value of the current cell.

So, a query might be `Q?=@<0`, *Is the current function value negative?*, or `Q?={@>=pi}`, *Is the current function value greater than or equal to* π*?* (The

---

[2] Documentation about `l3fp` can be found in `interface3.pdf`, which is part of the `l3kernel` bundle.

braces hide the equality sign.) `Q?={@=MIN}` (again note the braces) is the question: *Is the current function value equal to the minimum function value for the whole table?*

The answer must be in the form of a LaTeX 2$_\varepsilon$ formatting statement, again using `@` to denote the function value of the current cell. Thus `A!=\mathbf{@}` is a valid answer; so is `A!=\color{red}{@}` (provided you have `\usepackage{color}` in the preamble); and so is `A!=(@)`. Another valid answer is `A!=` , meaning that function values satisfying the `Q?` question are omitted from the output.

This can be useful to suppress 'irrelevant' values in particular contexts. For example, the *non-zero* values of `\binom{n}{m}`, displaying as $\binom{n}{m}$, are the ones of interest. Rather than cluttering the table with 0s, suppress them, first by finding them (`Q?={@=0}`), and then by excising them (`A!=`). The table suppresses display of the header row with the `headless` setting and instead displays it in the footer with the `foot=''` setting, automatically changing what would have been $n\backslash m$ in the header to n/m in the footer.

```
\tabulate[rspec={n,1,5},rhnudge=-18,rfont=bf,chfont=bf,
          cspec={m,1,5},ctitle=\binom nm\qquad,
          headless,rules=f',foot='',Q?={@=0},A!=]
  { \binom{n}{m} }[n=1,m=1]
```

$\Longrightarrow$

		$\binom{n}{m}$			
1	1				
2	2	1			
3	3	3	1		
4	4	6	4	1	
5	5	10	10	5	1
n/m	**1**	**2**	**3**	**4**	**5**

2.5.6.1 Star option: `\nmcTabulate*`

If the `Q?` question is satisfied by at least one function value then adding a star (asterisk) to the `\tabulate` command will display the first such instance. Like other starred commands in the `numerica` suite (`\eval*`, `info*`, `\macros*`, `\constants*`, `\iter*`, `\solve*` and `\recur*`), `\tabulate*` outputs a single number. Using the star means you do not need an answering `A!` to the query `Q?` since no formatting of table values is involved.

```
\tabulate*[rspec={n,1,12},cspec={m,1,4)},
           Q?={@<-1e-14||@>0.5+1e-14}]
  { \cos(m\pi/n) }[n=4,m=2][*4]
```

$\Longrightarrow$ 0.6235. Indeed, if you omit the `Q?` setting from the previous table so that all function values are visible then this is the value that follows 0.5000 in the `m=2` column, the first function value encountered outside the interval $[0, 0.5]$. (The `1e-14` is to ensure rounding errors don't give a spurious result.)

If you want the *maximum* value that has been tabulated then, from version 3 of `numerica-tables`, you do not even need the query: when `\tabulate` is starred, `Q?` is initialized behind the scenes to `@=MAX`. Thus, repeating the example from §2.2, a visual check of the table shows that the starred command has, indeed, isolated the maximum value displayed:

```
\tabulate[rspec={x,0.2,6},cspec={k,2,4},chnudge=27]
  { \sin kx }[k=3,x=0] $\longrightarrow
\tabulate*[rspec={x,0.2,6},cspec={k,2,4},chnudge=27]
  { \sin kx }[k=3,x=0] $
```

⟹

$x\backslash k$	3	5	7	9
0.2	0.564642	0.841471	0.985450	0.973848
0.4	0.932039	0.909297	0.334988	−0.442520
0.6	0.973848	0.141120	−0.871576	−0.772764
0.8	0.675463	−0.756802	−0.631267	0.793668
1.0	0.141120	−0.958924	0.656987	0.412118
1.2	−0.442520	−0.279415	0.854599	−0.980936

⟶ 0.98545

Package option There is a package option `Q?*` available which can be set to give some other default query than `@=MAX` if you so wish. For instance

```
\usepackage[Q?*={@=MIN}]{numerica-tables}
```

makes 'what is the minimum value tabulated?' the default query for a `\tabulate*` command. Or, `Q?*=@>0` makes 'what is the first positive value encountered?' the default query for a `\tabulate*` command, and so on.

Errors If *no* function value satisfies a query then a message is generated:

```
\tabulate*[rspec={n,1,12)},cspec={m,1,4},Q?=@>1]
  { \cos(m\pi/n) }[n=4,m=2][*4]
```

⟹ !!! No table value satisfies query `Q?` in: settings. !!!

Note that the same message is presented if the error arises from the package option query, which may well be confusing (for a moment) since there is no explicit query present in the settings option.

Scientific notation If you want the number output in scientific notation when the star option is chosen, then enter the exponent mark in the trailing number-format option of the `\tabulate*` command. This is straightforward for a mark like the commonly used letter `e`, but remember that if you enter `x` you will need to place the `\tabulate*` command between math delimiters, otherwise the `\times` symbol resulting from the `x` option will generate a LaTeX error ('Missing $ inserted'). Remembering the `$` signs gives, e.g.,

```
$ \tabulate*[rspec={n,1,12},cspec={m,1,4},
             Q?={@<-1e-14||@>0.5+1e-14}]
    { \cos(m\pi/n) }[n=4,m=2][*4x] $
```

⟹ 6.2349×10^{-1}.

2.6 Other matters

Here I group items that do not fit naturally into the earlier categories.

2.6.1 Nesting

A `\tabulate` command can be nested within other commands from the `numerica` suite, and those other commands can be nested within a `\tabulate` command.

Occasionally one might want to extract a value from a table to insert in another command. This can be done by nesting a `\tabulate*` command with an appropriate `Q?` setting within the other command. In fact, from version 2 of `numerica` on, the star is unnecessary. All we require is that the `Q?` setting is satisfied by at least one tabulated function value.

```
\eval[env=$]{(\tabulate
  [rspec={n,1,15},cspec={m,1,5},Q?={@=MAX}]
    { \cos(m\pi/n) }[n=4,m=2][*4])\sinh t +
      (\tabulate[rspec={n,1,15},cspec={m,1,5},Q?={@=MIN}]
         { \sin(m\pi/n) }[n=4,m=2][*4])\cosh t }
            [t=2][4]
```

$\Longrightarrow$ $(0.9397)\sinh t + (\,-1.0000)\cosh t = -0.354, \quad (t = 2)$.

Forming the table

```
\tabulate[rspec={n,1,15,0},rpos=2,rules=Tth,
          cspec={m,1,5},ctitle=*,chstyle=2]
  { \cos(m\pi/n) }[n=4,m=2][*4]
```

for the cosine (not presented here) and the table

```
\tabulate[rspec={n,1,15,0},rpos=2,rules=Tth,
          cspec={m,1,5},chstyle=2,ctitle=*]
  { \sin(m\pi/n) }[n=4,m=2][*4]
```

for the sine (also not presented here) and checking the entries shows that indeed the maximum and minimum values are 0.9397 and −1.0000 respectively.

If the `Q?` setting is not satisfied by any function value a familiar error message is shown – with a tweak:

```
\eval{$ (\tabulate
  [rspec={n,1,15},cspec={m,1,5},Q?=@>2]
     { \cos(m\pi/n) }[n=4,m=2][*4])\sinh t
      $}[t=2][4]
```

$\Longrightarrow$!!! No table value satisfies query `Q?` in: settings (2). !!!

Here, the `(2)` tells us that the message refers to a command at the second level, a *nested* command.

A more likely situation is to want to nest other commands within a `\tabulate` command. I give an example in the documentation to the associated package `numerica-plus` about timing of signals between points fixed on a rotating disk.

2.6.2 Saving tables to file

In earlier versions of `numerica-tables` it was possible to save a table to file, or a row or a column or a particular value from a table, by giving a *setting* `reuse` a value. From version 3.0.0, in the interests of simplifying use (and avoiding code complications) the `reuse` *setting* has been discontinued. The `\reuse` (or `\nmcReuse`) *command* remains (as part of the `numerica` package) and can be used to save the most recent table to file.

In the following example, a table is created and then saved to file and to the macro `\mytable` by the subsequent `\reuse` command:

```
\tabulate[rspec={x,0.25,5,2},rhnudge=9,rules=TthB,
          cspec={k,0.25,3,2},chstyle=2,ctitle=**]
  { a\sin kx }[a=2/\pi,{k}=3,{x}=0.25][*]
\reuse[renew]{mytable}
```

$\Longrightarrow$

	$a\sin kx,\quad (a=2/\pi)$		
x	$k=3.00$	$k=3.25$	$k=3.50$
0.00	0.000000	0.000000	0.000000
0.25	0.433945	0.462191	0.488633
0.50	0.635025	0.635685	0.626425
0.75	0.495337	0.412111	0.314439
1.00	0.089840	-0.068879	-0.223316

Now test the content of the control sequence

`\mytable` $\Longrightarrow$

	$a\sin kx,\quad (a=2/\pi)$		
x	$k=3.00$	$k=3.25$	$k=3.50$
0.00	0.000000	0.000000	0.000000
0.25	0.433945	0.462191	0.488633
0.50	0.635025	0.635685	0.626425
0.75	0.495337	0.412111	0.314439
1.00	0.089840	-0.068879	-0.223316

Yes, `\mytable` contains the table. The macro and its contents have also been saved to file, the `.nmc` file of the current document, hence `numerica-tables.nmc` in the present instance. The contents of this file can be edited in a text editor, or some limited file operations are possible with the `\reuse` command. These have been described in the associated document `numerica.pdf`.

2.6.3 Viewing the LaTeX form

In previous versions of `numerica-tables` the `dbg` and `view` settings were disabled. In version 3, they have been enabled to the extent that the LaTeX form of a table can be viewed by entering either `dbg=11` or, less nerdishly, `view` into the settings option of `\nmcTabulate`. In the example a familiar table is specified and built in LaTeX and its form displayed with the `view` setting:

```
\tabulate[view,rspec={x,0.2,5}]
  { \sin x }[x=0.2]
```

$\Longrightarrow$

LaTeX: \begin {tabular}[m]{rr}\toprule $x\mkern 0mu$&$\textstyle \sin x\mkern 0mu$\\ \midrule ${0.2}$&0.198669\\ ${0.4}$&0.389418\\ ${0.6}$&0.564642\\ ${0.8}$&0.717356\\ ${1}$&0.841471\\ \bottomrule \end {tabular}

Chapter 3

Reference summary

3.1 Commands defined in `numerica-tables`

`\nmcTabulate`, `\tabulate`

Package options

- default rule configuration: `rules`, `norules`; see §§1.1, 2.4.5.
- default query for `\tabulate*`: `Q?*`; see §2.5.6.1.

3.2 Settings for `\nmcTabulate`

Row variable specification: uniform case

See §2.1.1.

key	type	meaning	comment
`rvar`	token(s)	row variable	
`rstep`	real num.	step size	
`rstop`	real num.	stop value	excludes `rows`
`rows`	int	number of rows	excludes `rstop`
`rspec`	comma list	`{rvar`, `rstep`, `rows}`	short form spec.
`rround`	int	rounding	default: `1`

Row variable specification: non-uniform case

See §2.1.2.

key	type	meaning	comment
`rfunc`	token(s)	formula for row var. values	
`rdata`	comma list or macro	list or macro (containing list) of row var. values	
`rfile`	chars	filepath/filename	file contains list of row var. values
`rdelim`	char	item separator for `rdata` or `rfile` lists	defaults to `,` or `;` depending as `.` or `,` is decimal mark
`rverb`	fp (`0/0.5/1`)	display `rdata`, `rfile` values verbatim (`1`), or slash fractions formatted (`0.5`)	initialized to `0`

Row variable column formatting

See §2.1.3.

key	type	meaning	initial
`rpos`	int (`0`. . .`4`)	column placement	`1`
`ralign`	char (`r/c/l`)	horizontal alignment	`r`
`rfont`	chars	font (`\math<chars>`)	
`rhead`	tokens	header	
`rhnudge`	fp	nudge header `<fp>` mu	`0`
`rhsize`	int (`-4`. . .`5`)	font size relative to `0=\normalsize`	`0`
`rmath`	char (`s/t/d`)	script-, text-, displaystyle	`t`
`rvar'`	tokens	2nd row variable col. spec.	
`rhead'`	tokens	header of 2nd row var. col. (if it exists)	
`rhnudge'`	fp	nudge 2nd row var. col. header `<fp>` mu	`0`

Column-variable specification

See §2.2.

key	type	meaning	comment
`cvar`	token(s)	column variable	
`cstep`	real num.	step size	
`cstop`	real num.	stop value	either `cstop`
`cols`	int	number of columns	or `cols`
`cspec`	comma list	`{cvar,cstep,cols}`	short form spec.
`chround`	int	col. var. rounding	default: `0`

Column-variable header formatting

See §2.2.2.

key	type	meaning	default
`chstyle`	int (`0...4`)	header style	`0`
`chead`	token(s)	user-defined col. var. header	
`calign`	char (`r`/`c`/`l`)	column alignment	`r`
`chnudge`	fp	nudge header `chnudge` mu	`0`
`chfont`	chars	font (`\math<chars>`)	
`chmath`	char (`s`/`t`/`d`)	script-, text- or displaystyle	`t`
`chsize`	int (`-4...5`)	font size relative to `0 => \normalsize`	`0`

Function-value formatting

See §2.5.

key	type	meaning	initial
`(pad)`	int	t-notation phantom padding	
`signs`	int	sign handling for function values	`0`
`diffs`	int	insert differences, pre-pad with 0s	`0`
`round`	tokens	row/col. dependent rounding value	
`Q?`	tokens	special cell conditional	
`A!`	tokens	special cell formatting	

Whole-of-table formatting

See §2.4.

key	type	meaning	initial
`ctitle`	token(s)	collective title for columns	
`csubttl`	token(s)	subtitle row for function-value cols	
`calign`	char(`r`/`c`/`l`)	column alignment	`r`
`headless`		suppress header row	
`foot`	token(s)	table-wide footer row	
`rules`	char(s)	horizontal rule spec.	`ThB`
`norules`		cancel all rules	
`rpos`	int (`0`...`4`)	row variable col. position(s)	`1`
`rbloc`	integer comma list	row block specification	
`rblocsep`	length	extra space between row blocks	`1ex`
`transpose`		show funct. vals in rows	
`valign`	char (`t`/`m`/`b`)	vertical alignment of table relative to text baseline	`m`

Miscellaneous settings

- `view`, equivalent to `dbg=11`: show the LaTeX expression for the table; see §2.6.3.

Index