`numerica-basics`

Andrew Parsloe
(`ajparsloe@gmail.com`)

February 15, 2021

Abstract

The numerica package defines a command to wrap around a mathematical expression in its LaTeX form and, once values are assigned to variables, numerically evaluate it. The intent is to avoid the need to modify the LaTeX form of the expression being evaluated. For programs with a preview facility like LyX, or compile-as-you-go systems, interactive back-of-envelope calculations and numerical exploration are possible within the document being worked on. The package requires the bundles l3kernel and l3packages, and the amsmath and mathtools packages. Additional modules define commands to iterate and find fixed points of functions of a single variable, to find the zeros or extrema of such functions, to calculate the terms of recurrence relations, and to create multi-column tables of function values (which requires the booktabs package).

Note:

- This document applies to version 1.0.0 of numerica.sty.
- Reasonably recent versions of the LaTeX3 bundles l3kernel and l3packages are required.
- The package requires amsmath and mathtools.
- I refer many times in this document (especially §3.4) to *Handbook of Mathematical Functions*, edited by Milton Abramowitz and Irene A. Segun, Dover, 1965. This is abbreviated to *HMF*, often followed by a number like 1.2.3 to locate the actual expression referenced.

Contents

Chapter 1

Introduction

`numerica` is a LaTeX package offering the ability to numerically evaluate mathematical expressions in the LaTeX form in which they are typeset.[1]

There are a number of packages which can do calculations in LaTeX,[2] but those I am aware of all require the mathematical expressions they operate on to be changed to an appropriate syntax. Of these packages `xfp` comes closest to my objective with `numerica`. For instance, given a formula

```
\frac{\sin (3.5)}{2} + 2\cdot 10^{-3}
```

(in a math environment), this can be evaluated using `xfp` by transforming the expression to `sin(3.5)/2 + 2e-3` and wrapping this in the command `\fpeval`. In `numerica` you don't need to transform the formula, just wrap it in an `\eval` command (for the acutal calculation see §1.1.2):

```
\eval{ \frac{\sin (3.5)}{2} + 2\cdot 10^{-3} }.
```

`numerica`, like `xfp` and a number of other packages, uses `l3fp` (the LaTeX3 floating point module in `l3kernel`) as its calculational engine. To some extent the main command, `\nmcEvaluate`, short-name form `\eval`, is a pre-processor to `l3fp`, converting mathematical expressions written in the LaTeX form in which they will be typeset into an 'fp-ified' form that is digestible by `l3fp`. The aim is to make the command act as a wrapper around such formulas. Ideally, one should not have to make *any* adjustment to them, although any text on Fourier series suggests that hope in full generality is delusional. Surprisingly

[1] `numerica` evolved from the author's `calculyx` package that was designed for use with the document processor LyX and available for download from a link on the LyX wiki website (but not from CTAN).

[2] A simple search finds the venerable `calc` in the LaTeX base, `calculator` (including an associated `calculus` package), `fltpoint`, `fp` (*fixed* rather than floating point), `spreadtab` (using either `fp` or `l3fp` as its calculational engine) if you want simple spreadsheeting with your calculations, the elaborate `xint`, `pst-calculate` (a limited interface to `l3fp`), `l3fp` in the LaTeX3 kernel, and `xfp`, the LaTeX3 interface to `l3fp`. Other packages include a calculational element but are restricted in their scope. (`longdivision` for instance is elegant, but limited only to long division.)

often however it *is* possible. We will see shortly that even complicated formulas like

$$\cos\tfrac{m}{n}\pi-(1-4\sin^2\tfrac{m}{3n}\pi)\frac{\sin\tfrac{1}{n}\pi\sin\tfrac{m-1}{n}\pi}{2\sin^2\tfrac{m}{3n}\pi},$$

and

$$\left(\frac{1-4\sin^2\tfrac{m}{3n}\pi}{2\sin^2\tfrac{m}{3n}\pi}\right)\sin\tfrac{2m-3}{3n}\pi\sin\tfrac{m-3}{3n}\pi,$$

can be evaluated 'as is' (see below, §1.1.5). There is no need to shift the position of the superscript 2 on the sines, no need to parenthesize the arguments of sin and cos, no need to insert asterisks to indicate multiplication, no need to change the `\frac` and `\tfrac`-s to slashes, `/`, no need to delete the `\left` and `\right` that qualify the big parentheses (in the underlying LaTeX) in the second expression. Of course, if there are variables in an expression, as in these examples, they will need to be assigned values. And how the result of the evaluation is presented also requires specifying, but the aim is always: to evaluate mathematical expressions in LaTeX with as little adjustment as possible to the form in which they are typeset.

`numerica` is written in `expl3`, the programming language of the LaTeX3 project. It uses the LaTeX3 module `l3fp` (part of `l3kernel`) as its calculational engine. This enables floating point operations to 16 significant figures, with exponents ranging between -10000 and $+10000$. Many functions and operations are built-in to `l3fp` – arithmetic operations, trigonometric, exponential and logarithm functions, factorials, absolute value, max and min. Others have been constructed for `numerica` from `l3fp` ingredients – binomial coefficients, hyperbolic functions, sums and products – but to the user there should be no discernible difference.

Associated modules provide for additional operations: iteration, finding zeros, recurrence relations, mathematical table building. Further modules are planned (e.g. calculus).

1.1 How to use `numerica`

The package is invoked in the usual way:[3] put

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

in the LaTeX preamble. `numerica` requires the `amsmath` and `mathtools` packages and loads these automatically. `numerica` will also accept use of some relational symbols from the `amssymb` package provided that package is loaded; see §2.3.4.

[3] I use the angle-bracket notation to indicate optional user input. Of course what is input does not include the angle brackets.

1.1.1 Packages and package options

Version 1.0.0 of `numerica` has three package options.

`plus` By calling `numerica` with the `plus` package option,

`\usepackage[plus]{numerica}`

the file `numerica-plus.def` is loaded where a number of additional commands: `\nmcIterate`, `\nmcSolve`, `\nmcRecur` are defined. These enable the iteration of functions of a single variable[4], including finding fixed points; the solving of equations of the form $f(x) = 0$ (or the location of local maxima or minima); and the calculation of terms in recurrence relations (like the Fibonacci series or othogonal polynomials). See the associated document `numerica-plus.pdf`.

`tables` By calling `numerica` with the `tables` package option

`\usepackage[tables]{numerica}`

the file `numerica-tables.def` is loaded with with the command `\nmcTabulate` enabling the creation of multi-column tables of function values with a wide variety of formatting options (most of those employed in *HMF* in fact). See the associated document `numerica-tables.pdf`.

`lyx` By calling `numerica` with the `lyx` package option,

`\usepackage[lyx]{numerica}`

the file `numerica-lyx.def` is loaded with code enabling the full use of the `\nmcReuse` command in the document processor LyX (along with all other commands of the `numerica` package). Use of `numerica` in LyX exploits the mini-LaTeX runs of the instant preview facility of that program to give immediate feedback on calculations without requiring the whole document to be compiled. See Chapter 7.

More than one option can be used at a time by separating the options with a comma; e.g. `\usepackage[plus,tables]{numerica}`. However, apart from Chapter 7, the present document focuses on `numerica` when called with no options: `\usepackage{numerica}`.

1.1.2 Simple examples of use

A simple example of use is provided by the document

```
\documentclass{minimal}
\usepackage{numerica}
\begin{document}

    \eval{$ mc^2 $}[m=70,c=299 792 458][8x]

\end{document}
```

[4]At this stage!

We have a formula between math delimiters: `$ mc^2 $`. We have wrapped a command `\eval` around the lot, added an optional argument in parentheses specifying numericaal values for the quantities `m` and `c`, and concluded it all with a trailing optional argument specifying that the result should be presented to 8 places of decimals and in scientific notation (the `x`). Running `pdflatex` on this document generates a pdf displaying

$$mc^2 = 6.29128625 \times 10^{18}, \ \ (m = 70, c = 299792458)$$

where the formula (mc^2) is equated to the numerical value resulting from substituting the given values of m and c. Those values are displayed in a list following the result. The calculation is presented to 8 decimal places in scientific notation. (According to Einstein's famous equation $E = mc^2$ this is the enormous energy content, in joules, of what was once considered an average adult Caucasian male. Only a minute fraction is ever available.)

A second example is provided by the formula in earlier remarks:

```
\documentclass{minimal}
\usepackage{numerica}
\begin{document}

    \eval{\[ \frac{\sin(3.5)}{2} + 2\cdot 10^{-3} \]}

\end{document}
```

Running `pdflatex` on this document produces the result

$$\frac{\sin(3.5)}{2} + 2 \cdot 10^{-3} = -0.173392$$

The `\eval` command used in these examples is the main command of the `numerica` package. It is discussed in full in the next chapter, but first some preliminaries.

1.1.3 Display of the result

In what follows I shall write things like (but generally more complicated than)

`$ \eval{ 1+1 } $` $\Longrightarrow 2$

to mean: run `pdflatex` on a document containing `\eval{1+1}` in the document body to generate a pdf containing the calculated result (2 in this instance). In this case the `\eval` command is used *within* a math environment (delimited by the dollar signs). It is not limited to this behaviour. The command can also wrap *around* the math delimiters (as we saw in the previous examples):

`\eval{$ 1+1 $}` $\Longrightarrow 1 + 1 = 2.$

As you can see, the display that results is different.

- When the `\eval` command is used *within* a math environment, only the *result,* followed possibly by the *variable = value list* (see §2.2) is displayed.

Environments may include the various AMS environments as well as the standard LaTeX inline (`$ $`), `equation` (`\[ \]`) and `eqnarray` environments. For an example of `\eval` within an `align*` environment see §1.1.4 below.

- When the `\eval` command is wrapped *around* a math environment, the result is displayed in the form, *formula = result* (followed possibly by the *variable = value list*) within that environment,
  - If the formula is long or contains many variables then it may be desirable to split the display over two lines; see §2.2.3.3 and §3.1.10,

the whole presented as an inline expression if `$` delimiters are used, or as a display-style expression otherwise. (See the mc^2 example for an illustration.)

It is not clear to me that wrapping `\eval` *around* the AMS environments, except for `multline`, makes much sense, although it can be done. Here is an example of `\eval` wrapped around a `multline*` environment (the phantom is there so that the hanging + sign spaces correctly),

```
\eval{ \begin{multline*}
          1+2+3+4+5+6+7+8+9+10+\phantom{0}\\
            11+12+13+14+15+16+17+18+19
       \end{multline*} }
```

$\Longrightarrow$

$$\begin{multline*}1+2+3+4+5+6+7+8+9+10+\\ 11+12+13+14+15+16+17+18+19=190\end{multline*}$$

- It is also possible to dispense with math delimiters entirely, neither wrapped within nor wrapped around the `\eval` command, but in that case `numerica` acts as if `\eval` had been used within `\[` and `\]` and displays the result accordingly.

1.1.4 Exploring

When working on `numerica`'s predecessor package, I constantly tested it against known results to check for coding errors. One test was to ensure that

$$\left(1+\frac{1}{n}\right)^n$$

did indeed converge to the number e as n increased. Let's do that here. Try first $n = 10$:

`\eval{$ e-(1+1/n)^n $}[n=10][x]` $\Longrightarrow$
$e-(1+1/n)^n = 1.245394\times10^{-1}, \ (n=10).$

(The default number of decimal places displayed is 6.) The difference between e and $(1+1/n)^n$ is about an eighth (0.125) when $n=10$, which is encouraging but hardly decisive. The obvious thing to do is increase the value of n. I'll use an `align*` environment to 'prettify' the presentation of the results:

```
\begin{align*}
  e-(1+1/n)^{n} & =\eval{e-(1+1/n)^n}[n=1\times10^5][*x],\\
  e-(1+1/n)^{n} & =\eval{e-(1+1/n)^n}[n=1\times10^6][*x],\\
  e-(1+1/n)^{n} & =\eval{e-(1+1/n)^n}[n=1\times10^7][*x],\\
  e-(1+1/n)^{n} & =\eval{e-(1+1/n)^n}[n=1\times10^8][*x].
\end{align*}
```

(most of which was written using copy and paste) which produces

$$\begin{aligned}
e-(1+1/n)^{n} &= 1.359128\times10^{-5}, \qquad (n=1\times10^{5}),\\
e-(1+1/n)^{n} &= 1.359140\times10^{-6}, \qquad (n=1\times10^{6}),\\
e-(1+1/n)^{n} &= 1.359141\times10^{-7}, \qquad (n=1\times10^{7}),\\
e-(1+1/n)^{n} &= 1.359141\times10^{-8}, \qquad (n=1\times10^{8}).
\end{aligned}$$

Clearly $(1+1/n)^n$ converges to e, the difference between them being of order $1/n$, but that is not what catches the eye. There is an unanticipated regularity here. 1.35914? Double the number: `$\eval{2\times 1.35914}[5]$` $\Longrightarrow$ 2.71828 which is close enough to e to suggest a relationship, namely,

$$\lim_{n\to\infty} n\left(e-\left(1+\frac{1}{n}\right)^{n}\right)=\tfrac{1}{2}e.$$

This was new to me. Is it true? From the familiar expansion of the logarithm

$$\begin{aligned}
\ln\left(1+\frac{1}{n}\right)^{n} &= n\ln\left(1+\frac{1}{n}\right)\\
&= n\left(\frac{1}{n}-\frac{1}{2}\frac{1}{n^2}+\frac{1}{3}\frac{1}{n^3}-\dots\right)\\
&= 1-\frac{1}{2n}\left(1-\frac{2}{3}\frac{1}{n}+\frac{2}{4}\frac{1}{n^2}-\right)\\
&\equiv 1-\frac{1}{2n}E_n,
\end{aligned}$$

say. Since E_n is an alternating series and the magnitudes of the terms of the series tend to 0 monotonically, $1>E_n>1-2/3n$. From this and the inequalities $1/(1-x)>e^x>1+x$ when $x<1$ it proved a straightforward matter to verify the proposed limit.

1.1.5 Reassurance

In the course of some hobbyist investigations in plane hyperbolic geometry I derived the formula

$$\Phi_1(m,n)=\cos\tfrac{m}{n}\pi-(1-4\sin^{2}\tfrac{m}{3n}\pi)\frac{\sin\tfrac{1}{n}\pi\sin\tfrac{m-1}{n}\pi}{2\sin^{2}\tfrac{m}{3n}\pi},$$

for $m = 2, 3, \dots$ and integral $n \geq 2m + 1$. A key concern was: when is Φ_1 positive? After an embarrassingly laborious struggle, I managed to work this expression into the form

$$\Phi_2(m,n)=\left(\frac{1-4\sin^{2}\tfrac{m}{3n}\pi}{2\sin^{2}\tfrac{m}{3n}\pi}\right)\sin\tfrac{2m-3}{3n}\pi\sin\tfrac{m-3}{3n}\pi,$$

in which the conditions for positivity are clear: with $n \geq 2m + 1$, so that $m\pi/3n < \pi/6$, the first factor is always positive, the second is positive for $m \geq 2$, and the third is positive for $m \geq 4$. All well and good, but given the struggle to derive Φ_2, was I confident that Φ_1 and Φ_2 really are equal? It felt all too likely that I had made a mistake.

The simplest way to check was to see if the two expressions gave the same numericaal answers for a number of m, n values. I wrote `\eval{\[ \]}[m=2,n=5]` twice and between the delimiters pasted the already composed expressions for Φ_1 and Φ_2, namely:

```
\eval{\[
        \cos\tfrac{m}{n}\pi-(1-4\sin^{2}\tfrac{m}{3n}\pi)
        \frac{\sin\tfrac{1}{n}\pi\sin\tfrac{m-1}{n}\pi}
        {2\sin^{2}\tfrac{m}{3n}\pi}
      \]}[m=2,n=5]
\eval{\[
        \left(
          \frac{1-4\sin^{2}\tfrac{m}{3n}\pi}
          {2\sin^{2}\tfrac{m}{3n}\pi}
        \right)
        \sin\tfrac{2m-3}{3n}\pi\sin\tfrac{m-3}{3n}\pi
      \]}[m=2,n=5]
```

I have added some formatting – indenting, line breaks – to make the formulas more readable for the present document but otherwise left them unaltered. The `\eval` command can be used for even quite complicated expressions without needing to tinker with their LaTeX form, but you may wish – as here – to adjust white space to clarify the component parts of the formula. Running `pdflatex` on these expressions, the results were

$$\cos\tfrac{m}{n}\pi-(1-4\sin^{2}\tfrac{m}{3n}\pi)\frac{\sin\tfrac{1}{n}\pi\sin\tfrac{m-1}{n}\pi}{2\sin^{2}\tfrac{m}{3n}\pi}=-0.044193,\qquad(m=2,n=5)$$

$$\left(\frac{1-4\sin^2\frac{m}{3n}\pi}{2\sin^2\frac{m}{3n}\pi}\right)\sin\tfrac{2m-3}{3n}\pi\sin\tfrac{m-3}{3n}\pi = -0.044193, \qquad (m=2, n=5)$$

which was reassuring. Doing it again but with different values of m and n, again the results coincided:

$$\cos\tfrac{m}{n}\pi - (1-4\sin^2\tfrac{m}{3n}\pi)\frac{\sin\frac{1}{n}\pi\sin\frac{m-1}{n}\pi}{2\sin^2\frac{m}{3n}\pi} = 0.107546, \qquad (m=5, n=13)$$

$$\left(\frac{1-4\sin^2\frac{m}{3n}\pi}{2\sin^2\frac{m}{3n}\pi}\right)\sin\tfrac{2m-3}{3n}\pi\sin\tfrac{m-3}{3n}\pi = 0.107546, \qquad (m=5, n=13)$$

Thus reassured that there was *not* an error in my laborious derivation of Φ_2 from Φ_1, it was not difficult to work back from Φ_2 to Φ_1 then reverse the argument to find a straightforward derivation.

Chapter 2

\nmcEvaluate (\eval)

The main calculational command in `numerica` is `\nmcEvaluate`. Unlike some other commands which are loaded optionally, `\nmcEvaluate` is *always* loaded, and therefore always available. Because `\nmcEvaluate` would be tiresome to write too frequently, particularly for back-of-envelope calculations, there is an equivalent short-name form, `\eval`, used almost exclusively in the following. But note: wherever you see the command `\eval`, you can substitute `\nmcEvaluate` and obtain the same result.

`\eval` (like other short-name forms of other commands in the `numerica` suite) is defined using `\ProvideDocumentCommand` from the `xparse` package. Hence if `\eval` has already been defined in some other package already loaded, it will not be redefined by `numerica`. It will retain its meaning in the other package. Its consequent absence from `numerica` may be an irritant, but only that; `\nmcEvaluate` is defined using `xparse`'s `\DeclareDocumentCommand` which would override any (freakishly unlikely) previous definition of `\nmcEvaluate` in another package and would therefore still be available.

2.1 Syntax of \nmcEvaluate (\eval)

There are five arguments to the `\nmcEvaluate` (or `\eval`) command, of which only one, the third, is mandatory. All others are optional. If all are deployed the command looks like

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

I discuss the various arguments in the referenced sections.

1. `*` optional switch; if present ensures display of only the numerical result (suppresses display of the formula and vv-list); see §2.2.3.1

2. `[settings]` optional comma-separated list of *key=value* settings for this particular calculation; see §3.1

3. `{expr.}` the only mandatory argument; the mathematical expression/formula in LaTeX form that is to be evaluated
4. `[vv-list]` optional comma-separated list of *variable=value* items; see §2.2
5. `[num. format]` optional format specification for presentation of the numerical result (rounding, padding with zeros, scientific notation, boolean output); see §2.3

Note that arguments 4 and 5 are both square-bracket delimited optional arguments. Should only one such argument be used, `numerica` determines which is intended by looking for an equals sign within the argument. Its presence indicates the argument is the vv-list; its absence indicates the argument is the number format specification.

The vv-list and number-format specification are *trailing* optional arguments. There is a possibility that the `\eval` command could be followed by a square-bracketed mathematical expression which `numerica` might therefore confuse with one of its trailing arguments. Experience using `numerica` suggests that this will be a (very) rare occurrence and is easily prevented by inserting an empty brace pair (`{}`) before the offending square-bracketed expression. Allowing spaces between the arguments enables complicated expressions and large vv-lists to be formatted, in the interests of clarity, with new lines and white space – without requiring the insertion of comment characters (`%`).

Recommended practice is to minimise the number of optional arguments used in LaTeX commands by consolidating such arguments into a single *key=value* list. Although `numerica` uses such an argument, the vv-list does not fit naturally into that scheme. And practice suggests that separating out the elements of the number format specification (rounding value, padding with zeros, scientific notation, boolean output) and placing them in a trailing argument feels natural for the kind of back-of-envelope calculations envisaged for `numerica`.

2.2 The variable=value list

To evaluate algebraic, trigonometric and other formulas that involve *variables* we need to give those variables values. This is done in the *variable=value list* – or *vv-list* for short. This is the fourth argument of the `\nmcEvaluate` command and is a square-bracket delimited optional argument (optional because an expression may depend only on constants and numbers).

I was sorely tempted to use parentheses to delimit this argument, since then both the placement and delimiters of the vv-list would anticipate the way it is displayed in the evaluated result (see the mc^2 example in §1.1.2 above). But there is good reason not to. Parentheses will often occur in expressions in the vv-list. With parentheses nested within parentheses it is all too easy to get a pairing wrong, which would cause a LaTeX error and halt compilation. As it is, using the standard square bracket delimiters, unbalanced parentheses cause a `numerica` error (see §2.5.1), which does not halt compilation. (Of course unbalanced *square* brackets now will cause a LaTeX error, but such brackets are used less often in mathematical expressions and are rarely nested within other square-bracketed expressions.)

2.2.1 Variable names

In mathematical practice, variable names are generally single letters of the Roman or Greek alphabets, sometimes also from other alphabets, in a variety of fonts, and often with subscripts or primes or other decorations. In `numerica` a variable name is *what lies to the left of the equals sign in an item* of the vv-list. Thus variables can be multi-token affairs: x', x'', x^{iv}, x_n, x'_n, x''_{mn}, ${}^kC_n$, var, var, $Fred$, **Fred**, $\mathcal{FRED}$ ... Although variable names start and end with non-space tokens, a variable name may contain spaces – for instance `x x` should not cause a `numerica` error, but such names are not part of mathematical practice. Usually, for the kind of back-of-envelope calculations envisaged for `numerica`, and for ease of typing, most variables will be single letters from the Roman or Greek alphabets.

Because equals signs and commas give structure to the vv-list, it should also be clear that a variable name should not contain a *naked* equals sign or a *naked* comma. They can be incorporated in a variable name but only when decently wrapped in braces, like `R_{=}` displaying as $R_=$ or `X_{,i}` displaying as $X_{,i}$.

Note that x and x will be treated by `numerica` as *different* variables since, in the underlying LaTeX, one is `x` and the other `\mathrm{x}`. Even names that look identical in the pdf may well be distinct in LaTeX. This is true particularly of superscripts and subscripts: `x_0` and `x_{0}` appear identical in the pdf but in the underlying LaTeX they are distinct, and will be treated as distinct variables by `numerica`.

Although multi-token variables are perfectly acceptable, *internally* `numerica` expects variables to be single tokens. Hence a necessary initial step for the package is to convert all multi-token variable names in the vv-list and the formula to single tokens. `numerica` does this by turning the multi-token variable names into control sequences with names in the sequence `\_nmca`, `\_nmcb`, `\_nmcc`, etc., then searches through the vv-list and the formula for every occurrence of the multi-token names and replaces them with the relevant control sequences. It does this in order of decreasing size of name, working from the names that contain most tokens down to names containing only two tokens.

The conversion process uses computer resources. Even if there are no multi-token variables present, `numerica` still needs to check that this is so – unless the user alerts the program to the fact. This can be done by making a brief entry `xx=0` in the settings option (the second optional argument of `\nmcEvaluate`); see §3.1.4. If the user never (or hardly ever) uses multi-token variables, then a more permanent solution is to create a file `numerica.cfg` with the line `multitoken-variables = false`; see §3.3 for this.

2.2.2 The vv-list and its use

A vv-list is a comma-separated list where each item is of the form *variable=value*. It might be something simple like

```
[g=9.81,t=2]
```

or something more complicated like

```
[V_S=\tfrac43\pi r^3,V_C=2\pi r^2h,h=3/2,r=2].
```

Spaces around the equals signs or the commas are stripped away during processing so that

`[g=9.81,t=2]` and `[ g = 9.81 , t = 2]`

are the *same* variable=value list.

2.2.2.1 Evaluation from right to left

In these examples, with variables depending on other variables, there is an implication: that the list is evaluated *from the right*. Recall how a function of a function is evaluated, say $y = f(g(h(x)))$. To evaluate y, first x is assigned a value then $h(x)$ is calculated, then $g(h(x))$ then $f(g(h(x))) = y$. We work from right to left, from the innermost to the outermost element. Or consider an example like calculating the area of a triangle by means of the formula

$$A = \sqrt{s(s-a)(s-b)(s-c)}.$$

First we write the formula; then we state how s depends on a, b, c, namely $s = \frac{1}{2}(a+b+c)$, then we give values to a, b, c. In `numerica` this is mirrored in the layout of the `\eval` command:

```
\eval{$ \sqrt{s(s-a)(s-b)(s-c)} $}
              [s=\tfrac12(a+b+c),a=3,b=4,c=5]
```

The formula in a sense is the leftmost extension of the vv-list. The entire evaluation occurs from right to left.

This means that the rightmost variable in the vv-list can depend only on (mathematical) constants and numbers – although it may be a complicated expression of those elements. Other variables in the vv-list can depend on variables *to their right* but not to their left.

2.2.2.2 Expressions in the variable=value list

Suppose our expression is $\frac{4}{3}\pi r^3$, the volume V_S of a sphere in terms of its radius r, and we want to calculate the volume for different values of r to get a sense of how rapidly volume increases with radius.

`$ V_S=\eval{ \tfrac43\pi r^3 }[r=1] $` $\Longrightarrow V_S = 4.18879, \;\; (r = 1).$

Having set up this calculation it is now an easy matter to change the value of r in the vv-list:

`$ V_S=\eval{ \tfrac43\pi r^3 }[r=1.5] $` $\Longrightarrow V_S = 14.137167, \;\; (r = 1.5).$

`$ V_S=\eval{ \tfrac43\pi r^3 }[r=2] $` $\Longrightarrow V_S = 33.510322, \;\; (r = 2).$

To compute the volume $V_C = \pi r^2 h$ of a cylinder, we have two variables to assign values to:

`$ V_C=\eval{ \pi r^2h }[h=4/3,r=1] $` $\Longrightarrow$
$V_C = 4.18879, \;\; (h = 4/3, r = 1).$

Although values in the vv-list are generally either numbers or simple expressions (like `4/3`), that is not essential. A little more complicated is

`$ V_C=\eval{ hA_C }[A_C=\pi r^2,h=4/3,r=1] $` $\Longrightarrow$
$V_C = 4.18879, \;\; (A_C = \pi r^2, h = 4/3, r = 1).$

where calculation of the volume of the cylinder has been split into two: first calculate the area A_C of its circular base and then, once that has been effected, calculate the volume.

A second example is provided by Brahmagupta's formula for the area of a triangle in terms of its semi-perimeter. In a triangle ABC, the sides are $a = 3$, $b = 4$ and $c = 5$. (Of course we know this is a right-angled triangle with area $\frac{1}{2}ab = 6$.) The semi-perimeter $s = \frac{1}{2}(a + b + c)$ and the area of ABC is

```
    \eval{$ \sqrt{s(s-a)(s-b)(s-c) $}
                  [s=\tfrac12(a+b+c),a=3,b=4,c=5]
```

$\Longrightarrow \sqrt{s(s-a)(s-b)(s-c)} = 6, \;\; (s = \frac{1}{2}(a + b + c), a = 3, b = 4, c = 5).$

2.2.2.3 Constants

There are five constants built-in to `numerica`: `\pi`, the ratio of circumference to diameter of a circle, `e`, the base of natural logarithms, `\gamma`, the limit of $\left(\sum_1^N 1/n\right) - \ln N$ as $N \to \infty$, `\phi`, the golden ratio, equal to $\frac{1}{2}(1 + \sqrt{5})$, and the utilitarian constant `\deg`, the number of radians in a degree.

`\eval{$ \pi $}` $\Longrightarrow \pi = 3.141593,$

`\eval{$ e $}` $\Longrightarrow e = 2.718282,$

`\eval{$ \gamma $}` $\Longrightarrow \gamma = 0.577216,$
`\eval{$ \phi $}` $\Longrightarrow \phi = 1.618034,$
`\eval{$ \deg $}` $\Longrightarrow \deg = 0.017453,$

so that `\eval{$ 180\deg $}` $\Longrightarrow 180\deg = 3.141593$ (as it should).

Let's combine two of these in a formula:

`\eval{$ e^\pi-\pi^e $}` $\Longrightarrow e^\pi - \pi^e = 0.681535,$

which is close-ish to $\frac14 e$: `\eval{$ \tfrac14e $}` $\Longrightarrow \frac14 e = 0.67957.$

2.2.2.4 Use of `\pi`, `e`, `\gamma`, `\phi` as variables

In some contexts it may feel natural to use any or all of `\pi`, `e`, `\gamma` and `\phi` as variables by assigning values to them in the vv-list. `numerica` does not object. The values assigned in this way override the constants' values. For example, if the triangle we labelled ABC previously was instead labelled CDE then it has sides $c = 3, d = 4$ and (note!) $e = 5$. It's area therefore is

```
\eval{$ \sqrt{s(s-c)(s-d)(s-e)} $}
          [s=\tfrac12(c+d+e),c=3,d=4,e=5]
```

$\Longrightarrow$
$\sqrt{s(s-c)(s-d)(s-e)} = 6, \quad (s = \frac12(c+d+e), c = 3, d = 4, e = 5).$

Since this is the correct area we see that `e` has been treated as a variable with the assigned value 5, not as the constant. But if `e` (or `\pi` or `\gamma` or `\phi`) is not assigned a value in the vv-list then it has, by default, the value of the constant.

In the case of `e`, if you wish to use it as a variable, the constant is always available as `\exp(1)`. There is no similar alternative available for `\pi`, `\gamma` or `\phi` although you can always do something like `[\pi=<new value>,\pi_0=\pi]` in the vv-list, so that `\pi_0` now has the constant's value.

2.2.3 Display of the vv-list

By default, the vv-list is displayed with (in fact following) the numerical result. That and the format of the display can both be changed.

2.2.3.1 Star option: suppressing display of the vv-list

If display of the vv-list is not wanted at all, only the numerical result, it suffices to attach an asterisk (star) to the `\eval` command:

`$ V_C=\eval*{ hA_C }[A_C=\pi r^2,h=4/3,r=1] $` $\Longrightarrow V_C = 4.18879,$

or simply the naked result:

`\eval*{$ hA_C $}[A_C=\pi r^2,h=4/3,r=1]` $\Longrightarrow 4.18879.$

In the latter case, note that a negative result will display with a hyphen for the minus sign unless you, the user, explicitly write math delimiters around the `\eval*` command:

`\eval*{$ y $}[y=ax+b,x=2,a=-2,b=2]` $\Longrightarrow$ -2

The `$` signs that `\eval*` wraps around are ignored. The star option delivers a number, pure and simple, with no accompaniments.

2.2.3.2 Suppressing display of items

You may wish to retain some variables in the vv-list display, but not all. For those variables you wish omitted from the display, wrap each variable (but not the equals sign or value) in braces. When calculating the volume of a cylinder in the previous examples, the base area A_C has a different status from the 'fundamental' variables r and h. It is an intermediate value, one that we pass through on the way to the final result. To suppress it from display enclose the variable in braces:

`$ V_C=\eval{ hA_C }[{A_C}=\pi r^2,h=4/3,r=1] $` $\Longrightarrow$

$$V_C = 4.18879,\ \ (h = 4/3, r = 1).$$

As you can see, A_C no longer appears in the displayed vv-list. Of course the name and its value are still recorded 'behind the scenes' and can still be used in calculations.

2.2.3.3 Changing the display format

In two examples above, we have calculated the area of a triangle using Brahmagupta's formula. Display of the result is crowded. Two remedies have just been suggested, but a third one and preferable in this case would be to force display of the vv-list and result to a new line. This can be done through the settings option to the `\eval` command, discussed in §3.1.10. However, if `\eval` is wrapped around an *appropriate* environment (like `multline`, but not `equation`) it can also be done simply by including `\\` at the end of the formula.

In the following example I use Brahmagupta's formula for calculating the area of a cyclic quadrilateral (of which his formula for a triangle is a special case). The cyclic quadrilateral in the example is formed by a 45-45-90 triangle of hypotenuse 2 joined along the hypotenuse to a 30-60-90 triangle. The sides are therefore $\sqrt{2}, \sqrt{2}, \sqrt{3}, 1$. Adding the areas of the two triangles, the area of the quadrilateral is $A = 1 + \tfrac{1}{2}\sqrt{3}$, or in decimal form, `$\eval{1+\tfrac12\surd3}$` $\Longrightarrow$ 1.866025. Let's check with Brahmagupta's formula:

```
\eval{
  \begin{multline*}
    \sqrt{(s-a)(s-b)(s-c)(s-d)}\\
  \end{multline*}
     }[s=\tfrac12(a+b+c+d),
       a=\surd2,b=\surd2,c=\surd3,d=1]
```

$\Longrightarrow$

$$\sqrt{(s-a)(s-b)(s-c)(s-d)}$$
$$= 1.866025, \qquad (s = \tfrac{1}{2}(a+b+c+d), a = \sqrt{2}, b = \sqrt{2}, c = \sqrt{3}, d = 1)$$

2.3 Formatting the numerical result

A result of a calculation is displayed, by default, to 6 decimal places. All our results so far have been rounded to this figure, although not all digits are displayed, for instance if the sixth one is 0, or the result is an integer. Like other elements of the display, both rounding value and the (dis)appearance of trailing zeros can be customized, in this case by means of an optional argument following the vv-list (or the formula if there is no vv-list). This optional argument may contain up to four juxtaposed items from seven possibilities:

- a question mark ?, which gives boolean output, or
- an integer, the *rounding value*, positive, negative or zero, specifying how many decimal places to display the result to, or
- an asterisk *, which pads the result with zeros should it not have as many decimal places as the rounding value specifies, or
- the character `x` (lower case!) which presents the result in 'proper' scientific notation (a form like 1.234×10^5 for 123450), or
- the character `t` (lower case!) which presents the result in a bastardized form of scientific notation useful in tables (a form like (5)1.234 for 123450), or
- a character other than `?`, `*`, `x`, `t` or a digit, usually (but not necessarily) one of the letters `e d E D`, which presents the result in scientific notation using that character as the exponent mark (a form like 1.234e5 for 123450), or
- a prime
  - attached to the character specifying scientific notation, which extends that notation to numbers in the interval `[1,10)`, or
  - attached to a question mark, which changes the format of boolean output.

If you use `?` in the same specification as some other text character, the `?` prevails; if you use `x` in the same specification as some other text character except for `?`, the `x` prevails; if you use `t` in the same specification as some other text character except for `?` or `x`, the `t` prevails.

2.3.1 Rounding value

The rounding value specifies the number of decimal places displayed:

`$ \eval{ 1/3 }[4] $` $\Longrightarrow$ 0.3333

The default rounding value is 6:

`$ \eval{ 35/3 } $` $\Longrightarrow$ 11.666667

Following the default behaviour in `l3fp`, 'ties' are rounded to the nearest *even* digit. Thus a number ending 55 has a 'choice' of rounding to 5 or 6 and rounds up to the even digit 6, and a number ending 65 with a 'choice' of rounding to 6 or 7 rounds down to the even digit 6:

`$ \eval{ 0.1234555 } $` $\Longrightarrow$ 0.123456
`$ \eval{ 0.1234565 } $` $\Longrightarrow$ 0.123456

The calculational engine which `numerica` uses, `l3fp`, works to 16 significant figures and never displays more than that number (and often less).

- In the first of the following although I have specified a rounding value of 19 only 16 decimal places are displayed, with the final digit rounded up to 7;
- in the second I have added 10 zeros after the decimal point, meaning that all 19 decimal places specified by the rounding value can be displayed since the 10 initial zeros do not contribute to the significant figures;
- in the third I have changed the figure *before* the decimal point to 1 so that the 10 added zeros are now included among the significant figures;
- and in the fourth, I have added 9 digits before the decimal point:

`$ \eval{ 0.1234567890123456789 }[19] $` $\Longrightarrow$ 0.1234567890123457
`$ \eval{ 0.00000000001234567890123456789 }[19] $` $\Longrightarrow$
0.0000000000123456789
`$ \eval{ 1.00000000001234567890123456789 }[19] $` $\Longrightarrow$
1.000000000012346
`$ \eval{ 987654321.1234567890123456789 }[19] $` $\Longrightarrow$
987654321.1234568

In all cases, no more than 16 *significant* figures are displayed, although the number of decimal places displayed may exceed 16 as in the second example.

It is possible to use *negative* rounding values. Such a value zeroes the specified number of digits *before* the decimal point.

`$ \eval{ 987654321.123456789 }[-4] $` $\Longrightarrow$ 987650000

A rounding value of 0 rounds to the nearest integer:

`$ \eval{ 987654321.123456789 }[0] $` $\Longrightarrow$ 987654321

If you wish to change the *default* rounding value from 6 to some other value, this can be done by creating or editing a file `numerica.cfg` in a text editor; see §3.3.

2.3.2 Padding with zeros

A result may contain fewer decimal places than the rounding value specifies, the trailing zeros being suppressed by default (this is how `l3fp` does it). Sometimes, perhaps for reasons of presentation like aligning columns of figures, it may be desirable to pad results with zeros. This is achieved by inserting an asterisk, *, into the final optional argument of the `\eval` command:

`$ \eval{ 1/4 }[4] $` $\Longrightarrow$ 0.25,
`$ \eval{ 1/4 }[4*] $` $\Longrightarrow$ 0.2500.

2.3.3 Scientific notation

The `l3fp` package can output numbers in scientific notation. For example, 1234 is rendered as 1.234e3, denoting 1.234×10^3 , and 0.008 as 8e-3, denoting 8×10^{-3}. The 'e' here, the *exponent mark*, separates the *significand* (1.234) from the *exponent* (3). To switch on output in scientific notation in `numerica` enter `e` in the trailing optional argument:

`$ \eval{ 123456789 }[e] $` $\Longrightarrow$ 1.234568e8.

The default rounding value 6 is in play here. In `numerica`, when scientific notation is selected rounding takes a different meaning: it is the *significand* which is rounded (not the number as a whole). One digit precedes the decimal point, at most 15 follow it.

Negative rounding values are pointless for scientific notation. A zero might on occasion be relevant:

`$ \eval{ 987654321 }[0e] $` $\Longrightarrow$ 1e9.

Sometimes letters other than 'e' are used to indicate scientific notation, like 'E' or 'd' or 'D'. With a few exceptions, `numerica` allows any letter or text character to be used as the exponent marker:

`\eval{$ 1/23456789 $}[4d]` $\Longrightarrow$ $1/23456789 = 4.2632\text{d-}8$.

But when `x` is inserted in the trailing optional argument, the output is in the form $d_0.d_1 \ldots d_m \times 10^n$ (except when $n = 0$), where each d_i denotes a digit.

`\eval{$ 1/23456789 $}[4x]` $\Longrightarrow$ $1/23456789 = 4.2632 \times 10^{-8}$.

The requirements of tables leads to another form of scientific notation. Placing `t` in the trailing argument turns on this table-ready form of notation:

`\eval{$ 1/23456789 $}[4t]` $\Longrightarrow$ $1/23456789 = (-8)\,4.2632$.

This is discussed more fully in the associated document `numerica-tables.pdf`.

In the next example three options are used in the trailing argument. The order in which the items are entered does not matter:

`\eval{$ 1/125 $}[*e4]` $\Longrightarrow$ $1/125 = 8.0000$e-3.

Finally, to illustrate that 'any' text character[1] save for `x` or `t` can be used to distinguish the exponent, I use an @ character:

`\eval{$ 1/123 $}[@4]` $\Longrightarrow$ $1/123 = 8.1301$@-3.

2.3.3.1 Numbers in `[1,10)`

Usually when scientific notation is being used, numbers with magnitude in the interval $[1, 10)$ are rendered in their normal decimal form, 3.14159 and the like. Occasionally it may be desired to present numbers in this range in scientific notation (this can be the case in tables where the alignment of a column of figures might be affected). `numerica` offers a means of extending scientific notation to numbers in this range by adding a prime to the letter chosen as the exponent mark in the trailing optional argument.

`\eval{$ \pi $}[4t']` $\Longrightarrow$ $\pi = (0)\,3.1416$

2.3.3.2 \eval* and scientific notation

Scientific notation can be used for the numerical result output by `\eval*`:

`\eval*{$ \pi $}[e']` $\Longrightarrow$ 3.141593e0

There is one catch: if you substitute `x` for `e` here, LaTeX will complain about a missing `$`. An `x` in the number-format option produces a `\times` in the output which requires a math environment. It is up to you, as the user, to provide the necessary delimiters outside the `\eval*` command. (This applies even when `\eval*` wraps around math delimiters.)

(Because of the way `numerica` parses the number-format option, entering a prime with neither exponent character nor question mark specified will result in scientific output using `e` as the exponent mark. The last example could have been written `\eval*{$ \pi $}[']`.)

2.3.4 Boolean output

`l3fp` can evaluate comparisons, outputting 0 if the comparison is false, 1 if it is true. By entering a question mark, `?`, in the trailing optional argument, you can force `numerica` to do the same depending as the result of a calculation is

[1]Be sensible! An equals sign for instance might confuse `numerica` into thinking the number-format option is the vv-list, and will certainly confuse the reader.

zero or not. The expression being evaluated does not need to be a comparison, `$ \eval{\pi}[?] $` $\Longrightarrow$ 1, but comparisons are what this is designed for.

Possible comparison relations are `=`, `<`, `>`, `\ne`, `\neq`, `\ge`, `\geq`, `\le`, `\leq`. Although programming languages use combinations like `<=` or `>=`, `numerica` does *not* accept these (they are not part of standard *mathematical* usage) and will generate an error. An example where the relation is equality exhibits a numerological curiosity:[2]

`\eval[p=.]{\[ \frac1{0.0123456789}=81 \]}[5?]` $\Longrightarrow$

$$\frac{1}{0.0123456789}=81\to 1.$$

Notice the 5 alongside the question mark in the trailing argument. That is critical. Change the 5 to a 6 (or omit it since the default rounding value is 6) and the outcome is different:

`\eval[p=.]{\[ \frac1{0.0123456789}=81 \]}[6?]` $\Longrightarrow$

$$\frac{1}{0.0123456789}=81\to 0.$$

Now the relation is false. Evaluating the fraction to more than 6 places, say to 9, we can see what is going on:

`\eval{$ 1/0.0123456789 $}[9]` $\Longrightarrow$ $1/0.0123456789 = 81.000000737$.

2.3.4.1 Outputting `T` or `F`

To my eye, outputting 0 or 1 in response to a 'question' like $1/0.0123456789 = 81$ is confusing. It is easy to change the boolean output from $0, 1$ to a more appropriate F, T, or `F,T` by adding a prime or two primes respectively to the question mark in the number-format option.

`\eval[p=.]{\[ \frac1{0.0123456789}=81 \]}[6?'']` $\Longrightarrow$

$$\frac{1}{0.0123456789}=81\to \texttt{F}.$$

The default boolean output format is chosen to be $0, 1$ in case an `\eval*` command is used within another `\eval` command ('nesting'– see Chapter 4). The inner command needs to output a *numerical* answer.

2.3.4.2 Rounding error tolerance

If at least one of the terms in a comparison is the result of a calculation, then it's value is likely to contain rounding errors. What level of rounding error can

[2]The `[p=.]` of this and the next example ensures a full stop appears in the correct place; see §3.1.11.

we tolerate before such errors interfere with the comparison being made? `l3fp` tolerates none. It decides the truth or falsity of a comparison to all 16 significant figures: 1.000 0000 0000 0000 and 1.000 0000 0000 0001 are *not* equal in `l3fp`. But for most purposes this will be far too severe a criterion.

Suppose our comparison relation is ϱ, denoting one of $=$, $<$, $>$, `\le`, etc. If $X \varrho Y$ then $X - Y \varrho Y - Y$, i.e. $X - Y \varrho 0$. This is what `numerica` does. It takes the right-hand side of the relation from the left-hand side and then compares the *rounded* difference under ϱ to 0. The rounding value used is the number specified with the question mark in the trailing argument of the `\eval` command or, if no number is present, the default rounding value ('out of the box' this is 6). Thus, in a recent example, $1/0.0123456789 - 81$ when rounded to 5 decimal places is .0.00000, indistinguishable from zero at this rounding value; hence the equality $1/0.0123456789 = 81$ is true. But when rounded to 6 places it is 0.000001 which *is* distinguishable from zero and so the equality is false. Truth or falsity depends on the rounding value.

When dealing with numbers generated purely mathematically, rounding values of 5 or 6 are likely to be too small. More useful would be rounding values closer to `l3fp`'s 16 – perhaps 14? – depending on how severe the calculations are that generate the numbers. However if the numbers we are dealing with come from outside mathematics, from practical experiments perhaps, then even a rounding value of 5 or 6 may be too large.

2.3.4.3 Rationale

Mathematically, the claim that $X = Y$ at a rounding value n is the claim that

$$|X - Y| \leq 5 \times 10^{-(n+1)}.$$

since this rounds *down* to zero at n places of decimals. This gives a more accurate test of equality than doing things in the opposite order – rounding each number first and then taking the difference. One might, for instance, have numbers like $X = 0.12345$, $Y = 0.12335$. Rounding to $n = 4$ places, both round to 0.1234 and yet the difference between them is 0.0001 – they are distinguishable numbers to 4 places of decimals. This is why `numerica` forms the difference *before* doing the rounding.

2.3.4.4 And, Or, Not

For logical And LaTeX provides the symbols `\wedge` and `\land`, both displaying as $\wedge$, but `numerica` adds thin spaces (`\,`) around the symbol for `\land` (copying the package `gn-logic14.sty`). For logical Or LaTeX provides the symbols `\vee` and `\lor`, both displaying as $\vee$, but again `numerica` adds thin spaces around the symbol for `\lor`.

`\eval{$ 1<2 \wedge 2<3 $}[?'']` $\Longrightarrow 1 < 2 \wedge 2 < 3 \to$ `T`,

`\eval{$ 1<2 \land 2<3 $}[?'']` $\Longrightarrow 1 < 2 \,\wedge\, 2 < 3 \to$ `T`.

To my eye the second of these with its smidgen more space around the wedge symbol displays the meaning of the overall expression better than the first. Both And and Or have equal precedence; in cases of ambiguity the user needs to parenthesize as necessary to clarify what is intended.

LaTeX provides two commands for logical Not, `\neg` and `\lnot`, both displaying as $\neg$. Not binds tightly to its argument:

`\eval{$ \lnot A \land B $}[A=0,B=0]` $\Longrightarrow \neg A \,\wedge\, B = 0,\ \ (A = 0, B = 0).$

Here `\lnot` acts only on the A; if it had acted on $A \,\wedge\, B$ as a whole the result would have been 1.

For a little flourish, I evaluate a more complicated logical statement:[3]

```
\eval{$(A\lor\lnot C)\land(C\lor B)\land
            (\lnot A\lor\lnot B)$}[A=1,B=0,C=1][?'']
```

$\Longrightarrow (A \,\vee\, \neg C) \,\wedge\, (C \,\vee\, B) \,\wedge\, (\neg A \,\vee\, \neg B) \to \texttt{T},\ \ (A = 1, B = 0, C = 1)$

2.3.4.5 Chains of comparisons

`numerica` can handle chains of comparisons like $1 < 2 < 1 + 2 < 5 - 1$. 'Behind the scenes' it inserts logical And-s into the chain, $1 < 2 \wedge 2 < 1{+}2 \wedge 1{+}2 < 5{-}1$, and evaluates the modified expression:

`\eval{$ 1<2<1+2<5-1 $}[?'']` $\Longrightarrow 1 < 2 < 1 + 2 < 5 - 1 \to \texttt{T}.$

2.3.4.6 `amssymb` comparison symbols

`numerica` accepts some alternative symbols for the basic comparison relations from the `amssymb` package provided that package is loaded, i.e. the preamble of your document includes the statement

```
\usepackage{amssymb}
```

The variants from this package are: `\leqq` ($\leqq$), `\leqslant` ($\leqslant$), `\geqq` ($\geqq$), and `\geqslant` ($\geqslant$).[4] There are also negations: `\nless` ($\nless$), `\nleq` ($\nleq$), `\nleqq` ($\nleqq$), `\nleqslant` ($\nleqslant$), `\ngtr` ($\ngtr$), `\ngeq` ($\ngeq$), `\ngeqq` ($\ngeqq$), `\ngeqslant` ($\ngeqslant$).

[3]Quoting from an article in *Quanta Magazine* (August 2020) by Kevin Hartnett: 'Let's say you and two friends are planning a party. The three of you are trying to put together the guest list, but you have somewhat competing interests. Maybe you want to either invite Avery or exclude Kemba. One of your co-planners wants to invite Kemba or Brad or both of them. Your other co-planner, with an ax to grind, wants to leave off Avery or Brad or both of them. Given these constraints, you could ask: Is there a guest list that satisfies all three party planners?' I have written C for Kemba, A and B for Avery and Brad.

[4]No, that is not `eggplant`.

2.4 Calculational details

2.4.1 Arithmetic

Addition, subtraction, multiplication, division, square roots, n-th roots, and exponentiating (raising to a power) are all available.

Multiplication can be rendered explicitly with an asterisk,

`\eval{$ 9*9 $}` $\Longrightarrow 9*9=81,$

but that's ugly. More elegant is to use `\times`:

`\eval{$ 9\times9 $}` $\Longrightarrow 9\times 9=81.$

`\cdot` is also available and in many cases juxtaposition alone suffices:

`\eval{$ \surd2\surd2 $}` $\Longrightarrow \surd2\surd2=2,$

`\eval{$ ab $}[a=123,b=1/123]` $\Longrightarrow ab=1,\ \ (a=123,b=1/123).$

Division can be rendered in multiple ways too:

`\eval{$ 42/6 $}` $\Longrightarrow 42/6=7,$

`\eval{$ 42\div6 $}` $\Longrightarrow 42\div 6=7,$

or by using `\frac` or `\tfrac` or `\dfrac` as in

`\eval{$ \frac{42}6 $}` $\Longrightarrow \frac{42}{6}=7.$

But note that since juxtaposition means multiplication, it is also true that $42\frac{1}{6}$ evaluates to 7 inside an `\eval` command rather than denoting 'forty two and a sixth'. Hence if you want to use 'two and a half' and similar values in `numerica`, they need to be entered as improper fractions like $\frac{5}{2}$ or in decimal form, 2.5 (as one does automatically in mathematical expressions anyway because of the ambiguity in a form like $2\frac{1}{2}$).

2.4.1.1 Square roots and n-th roots

Let us check that 3, 4, 5 and 5, 12, 13 really are Pythagorean triples (I use `\sqrt` in the first, `\surd` in the second):

`\eval{$ \sqrt{3^2+4^2} $}` $\Longrightarrow \sqrt{3^2+4^2}=5,$

`\eval{$ \surd(5^2+12^2) $}` $\Longrightarrow \surd(5^2+12^2)=13.$

The `\sqrt` command has an optional argument which can be used for extracting n-th roots of a number. This notation is generally used when n is a small positive integer like 3 or 4. This practice is followed in `numerica`: n must be a (not necessarily small) *positive integer*:

`\eval{$ \sqrt[4]{81} $}` $\Longrightarrow \sqrt[4]{81}=3,$

`\eval{$ \sqrt[n]{125} $}[n=\floor{\pi}]` $\Longrightarrow \sqrt[n]{125}=5,\ \ (n=\lfloor\pi\rfloor).$

If n should not be a positive integer, an error message is generated; see §2.5.

For display-style expressions, the `\sqrt` command grows to accommodate the extra vertical height; the surd doesn't. Here is an example which anticipates a number of matters not discussed yet. It shows `\eval` wrapping around a square root containing various formatting commands (negative spaces, `\left` and `\right` nested within `\bigg` commands), all digested without complaint (see §2.4.4; and see §3.1.11 for the `[p=.]`):

```
\eval[p=.]
  {\[
    \sqrt[3]
      {\!\biggl(\!\left.\frac AD\right/\!\frac BC\biggr)}
  \]}[A=729,B=81,C=9,D=3]
```

$\Longrightarrow$

$$\sqrt[3]{\biggl(\left.\frac{A}{D}\right/\frac{B}{C}\biggr)} = 3, \qquad (A = 729, B = 81, C = 9, D = 3).$$

As implemented in `numerica`, n-th roots found using `\sqrt[n]` are `n=<integer>` roots. This raises an interesting question: if the 'n' of an n-th root is the result of a calculation, what happens with rounding errors? The calculation may not produce an *exact* integer. (This problem also arises with factorials; see §2.4.12.) The solution employed in `numerica` is to make what is considered an integer depend on a rounding value. Most calculations will produce rounding errors in distant decimal places. For 'int-ifying' calculations, `numerica` uses a rounding value of 14: a calculation produces an integer if, when rounded to 14 figures, the result is an integer. Since `l3fp` works to 16 significant figures, a rounding value of 14 allows ample 'elbowroom' for rounding errors to be accommodated when judging what is an integer and what is not. As a practical matter problems should not arise.

2.4.1.2 *n*-th roots of negative numbers

Odd (in the sense of 'not even') integral roots of *negative* numbers are available with `\sqrt`,

`\eval{$ \sqrt[3]{-125} $}` $\Longrightarrow \sqrt[3]{-125} = -5,$

`\eval{$ \sqrt[3]{-1.25} $}` $\Longrightarrow \sqrt[3]{-0.125} = -0.5.$

2.4.1.3 Inverse integer powers

Of course to find an n-th root we can also raise to the inverse power,

`\eval{$ 81^{1/4} $}` $\Longrightarrow 81^{1/4} = 3.$

However, raising a *negative* number to an inverse power generates an error even when, mathematically, it should not. This matter is discussed below in §2.5.4.

2.4.2 Precedence, parentheses

The usual precedence rules apply: multiplication and division bind equally strongly and more strongly than addition and subtraction which bind equally stongly. Exponentiating binds most strongly. Evaluation occurs from the left.

`\eval{$ 4+5\times6+3 $}` $\Longrightarrow 4+5\times 6+3=37$,

`\eval{$ 6\times10^3/2\times10^2 $}` $\Longrightarrow 6\times 10^3/2\times 10^2=300000$,

which may not be what was intended. Parentheses (or brackets or braces) retrieve the situation:

`\eval{$ (4+5)(6+3) $}` $\Longrightarrow (4+5)(6+3)=81$,

`\eval{$ (6\times10^3)/(2\times10^2) $}` $\Longrightarrow (6\times 10^3)/(2\times 10^2)=30$.

Because exponentiating binds most strongly, negative values must be parenthesized when raised to a power. If not,

`\eval{$ -4^2 $}` $\Longrightarrow -4^2=-16$,

which is clearly not $(-4)^2$. But

`\eval{$ (-4)^2 $}` $\Longrightarrow (-4)^2=16$.

2.4.2.1 Command-form brackets

Note that brackets of all three kinds are available also in command form: `\lparen \rparen` (from `mathtools`) for `( )`, `\lbrack \rbrack` for `[ ]`, and `\lbrace \rbrace` for `\{ \}`.

2.4.3 Modifiers (`\left \right`, etc.)

The `\left` and `\right` modifiers and also the series of `\big...` modifiers (`\bigl \bigr`, `\Bigl \Bigr`, `\biggl \biggr`, `\Biggl \Biggr`) are available for use with all brackets (parentheses, square brackets, braces):

```
\eval[p=.]{\[ \exp\left(
    \dfrac{\ln2}{4}+\dfrac{\ln8}{4}
  \right) \]}
```

$\Longrightarrow$

$$\exp\left(\frac{\ln 2}{4}+\frac{\ln 8}{4}\right)=2.$$

`numerica` also accepts their use with `.` (dot) and with `/` (as noted earlier, the `[p]` and `[p=.]` are explained at §3.1.11):

`\eval[p]{\[ \left.\dfrac{3+4}{2+1}\right/\!\dfrac{1+2}{4+5} \]}`

$\Longrightarrow$

$$\left.\frac{3+4}{2+1}\right/\!\frac{1+2}{4+5}=7.$$

They can be nested.

2.4.4 Other formatting commands

There are many formatting commands which change the layout of a formula on the page but do not alter its content. These include various spacing commands like `\!`, `\quad`, etc., phantoms (`\phantom` etc.), `\mathstrut` from TEX and its `mathtools` cousin `\xmathstrut`.

Consider the same package's `\splitfrac` and `\splitdfrac`. The `mathtools` documentation gives an example to illustrate the use of these last two. I've mangled it to produce a ridiculous illustration of their use, and of the modifiers `\left \right`, and of the command-form alternatives to parentheses `\lparen \rparen`; also the use of `\dfrac`. A little mental arithmetic will convince that we are evaluating the square root of $(9 \times 7)^2$ which indeed is what we get:[5]

```
\eval[p=.,vvd=]{\[
    \sqrt{\left\lparen
      \frac{ \splitfrac{xy + xy + xy + xy + xy}
              {+ xy + xy + xy + xy}
           }
           { \dfrac z7}
          \right\rparen \left\lparen
            \frac{ \splitdfrac{xy + xy + xy + xy + xy}
                     {+ xy + xy + xy + xy}
                 }
                 {\dfrac z7}\right\rparen}
      \]}[x=2,y=5,z=10]
```

$\Longrightarrow$

$$\sqrt{\left(\frac{\splitfrac{xy+xy+xy+xy+xy}{+xy+xy+xy+xy}}{\dfrac{z}{7}}\right)\left(\frac{\splitdfrac{xy+xy+xy+xy+xy}{+xy+xy+xy+xy}}{\dfrac{z}{7}}\right)}=63.$$

`numerica` essentially ignores formatting commands (the ones it knows of). They do not alter the mathematical content of a formula, only how it looks. But there will undoubtedly be formatting commands it does not recognize which will probably trigger an 'Unknown token' message. Please contact the author in that case.[6]

2.4.5 Trigonometric & hyperbolic functions

LATEX provides all six trignometric functions, `\sin`, `\cos`, `\tan`, `\csc`, `\sec`, `\cot` and the three principal inverses `\arcsin`, `\arccos`, `\arctan`. It also provides four of the six hyperbolic functions: `\sinh`, `\cosh`, `\tanh`, `\coth`, and

[5]For the `[p=.,vvd=]` see §3.1.11 and §3.1.9. The first puts the concluding full stop in the right place; the second suppresses the vv-list.

[6]ajparsloe@gmail.com

no inverses. `numerica` provides the missing hyperbolic functions, `\csch` and `\sech`, and all missing inverses, the three trigonometric and all six hyperbolic: `\arccsc`, `\arcsec`, `\arccot`, and `\asinh`, `\acosh`, `\atanh`, `\acsch`, `\asech`, `\acoth`. (*HMF* writes arcsinh, arccosh, etc. and ISO recommends arsinh, arcosh, etc. The first seems ill-advised, the second not widely adopted. At present neither is catered for in `numerica`.)

`\eval{$ \arctan1/1\deg $}` $\Longrightarrow \arctan 1/1\deg = 45$,
`\eval{$ \atanh\tanh3 $}` $\Longrightarrow \operatorname{atanh}\tanh 3 = 3$.

Inverses can also be constructed using the ‘-1’ superscript notation. Thus

`\eval{$ \sin^{-1}(1/\surd2)/1\deg $}` $\Longrightarrow \sin^{-1}(1/\surd 2)/1\deg = 45$,
`\eval{$ \tanh\tanh^{-1}0.5 $}` $\Longrightarrow \tanh\tanh^{-1}0.5 = 0.5$.

Hyperbolic functions

Please note that `l3fp` does not (as yet) provide *any* hyperbolic functions natively. The values `numerica` provides for these functions are *calculated* values using familiar formulas involving exponentials (for the direct functions) and natural logarithms and square roots for the inverses. Rounding errors mean the values calculated may not have 16-figure accuracy. The worst ‘offenders’ are likely to be the least used, `\acsch` and `\asech`. For instance,

$$\operatorname{acsch} x = \ln\left[\frac{1}{x} + \left(\frac{1}{x^2}+1\right)^{1/2}\right],$$

`\eval{$ \csch \acsch 7 $}[16]` $\Longrightarrow \operatorname{csch}\operatorname{acsch} 7 = 6.999999999999983.$

2.4.6 Logarithms

The natural logarithm `\ln`, base 10 logarithm `\lg`, and binary or base 2 logarithm `\lb` are all recognized, as is `\log`, preferably with a subscripted base:

`\eval{$ \log_{12}1728 $}` $\Longrightarrow \log_{12} 1728 = 3$

If there is no base indicated, base 10 is assumed. (The notations `\ln`, `\lg`, and `\lb` follow ISO 80000-2 recommendation, which frowns upon the use of the unsubscripted `\log` although only `\ln` appears widely used.) The base need not be explicitly entered as a number. It could be entered as an expression or be specified in the vv-list:

`\eval*{$ \log_b c $}[b=2,c=1024]` $\Longrightarrow 10$,

the log to base 2 in this case. It is possible to use the unadorned `\log` with a base different from 10; if you wish to do this only for a particular calculation see §3.1.7, or see §3.3 if you want to make this default behaviour.

2.4.7 Other unary functions

Other unary functions supported are the exponential function `\exp` and signature function `\sgn` (equal to -1, 0, or 1 depending as its argument is < 0, $= 0$, or > 0).

2.4.8 Squaring, cubing, ... unary functions

`numerica` has no difficulty reading a familiar but 'incorrectly formed' expression like

$$\sin^2 1.234 + \cos^2 1.234.$$

You do not have to render it $(\sin 1.234)^2 + (\cos 1.234)^2$ or (heaven forbid) $(\sin(1.234))^2 + (\cos(1.234))^2$. The everyday usage is fine:

`\eval{$ \sin^2\theta+\cos^2\theta $}[\theta=1.234]` $\Longrightarrow$

$$\sin^2\theta + \cos^2\theta = 1, \quad (\theta = 1.234)\ .$$

Equally `numerica` has no difficulty reading the 'correct' but pedantic form

`\eval{$ (\sin(\theta))^2+(\cos(\theta))^2 $}[\theta=1.234]` $\Longrightarrow$

$$(\sin(\theta))^2 + (\cos(\theta))^2 = 1, \quad (\theta = 1.234)\ .$$

A hyperbolic identity is confirmed in this example:

`\eval{$ \sinh 3x $}[x=1]` $\Longrightarrow \sinh 3x = 10.017875, \quad (x = 1),$

`\eval{$ 3\sinh x+4\sinh^3x $}[x=1]` $\Longrightarrow$

$$3\sinh x + 4\sinh^3 x = 10.017875, \quad (x = 1).$$

In fact all named unary functions in `numerica` can be squared, cubed, etc., in this 'incorrect' but familiar way, although the practice outside the trigonometric and hyperbolic context seems (vanishingly?) rare.

When the argument of the function is parenthesized and raised to a power – like $\sin(\pi)^2$ – it is read by `numerica` as the 'sine of the square of pi', $\sin(\pi^2)$, and *not* as the 'square of the sine of pi', $(\sin\pi)^2$:

`\eval{$ \sin(\pi)^2 $}` $\Longrightarrow \sin(\pi)^2 = -0.430301\ .$

Things are done like this in `numerica` above all to handle the logarithm in a natural way. Surely $\ln x^n = n \ln x = \ln(x^n)$ rather than $(\ln x)^n$? And if we wish to write (as we do) $\ln(1 + 1/n)^n = n\ln(1 + 1/n) = 1 - 1/2n + 1/3n^2 - \ldots$ to study the limiting behaviour of $(1 + 1/n)^n$, then we cannot avoid $\ln(x)^n = n\ln(x) = \ln(x^n)$.

2.4.9 *n*-ary functions

The functions of more than one variable (n-ary functions) that `numerica` supports are `\max`, `\min` and `\gcd`, greatest common divisor. The comma list of arguments to `\max`, `\min` or `\gcd` can be of arbitrary length. The arguments themselves can be expressions or numbers. For `\gcd`, non-integer arguments are truncated to integers. Hence both y and $3y$ are independently truncated in the following example – to 81 and 243 respectively:

`\eval{$ \gcd(12,10x^2,3y,y,63) $}[y=1/0.0123456789,x=3]` $\Longrightarrow$

$$\gcd(12, 10x^2, 3y, y, 63) = 3, \quad (y = 1/0.0123456789, x = 3)\ .$$

(The truncation occurs in the argument of `\gcd`, not in the vv-list.)

For n-ary functions, squaring, cubing, etc. follows a different pattern from that for unary functions. For `\max`, `\min`, `\gcd` the argument of the function is a comma list. Squaring the argument makes no sense. We understand the superscript as applying to the function as a whole. (Consistency is not the point here; it is what mathematicians do that `numerica` tries to accommodate.)

`\eval{$ \gcd(3x,x,\arcsin 1/\deg)^2 $}[x=24]` $\Longrightarrow$

$$\gcd(3x, x, \arcsin 1/\deg)^2 = 36, \quad (x = 24)\ .$$

2.4.10 Delimiting arguments with brackets & modifiers

Arguments of unary and n-ary functions can be delimited not only with parentheses, but also with square brackets and braces, both in explicit character form and also in the command form of §2.4.2.1. The brackets, of whatever kind, can be qualified with `\left \right`, `\bigl \bigr`, etc.[7]

`\eval[p=.]{\[ \sin\left\lbrack \dfrac\pi{1+2+3}\right\rbrack \]}`

$$\Longrightarrow$$

$$\sin\left[\frac{\pi}{1+2+3}\right] = 0.5.$$

2.4.11 Absolute value, floor & ceiling functions

It is tempting to use the `|` key on the keyboard for inserting an absolute value sign. `numerica` accepts this usage, but it is deprecated. The spacing is incorrect – compare $| - l|$ using `|` against $\lvert -l \rvert$ using `\lvert \rvert`. Also, the identity of the left and right delimiters makes nested absolute values difficult to parse. `numerica` does not attempt to do so. Placing an absolute value constructed with `|` within another absolute value constructed in the same way is likely to produce a compilation error or a spurious result. `\lvert \rvert` are better in every way except ease of writing. To aid such ease `numerica` provides the `\abs` function (using the `\DeclarePairedDelimiter` command of the `mathtools` package). This takes a mutually exclusive star (asterisk) or square bracketed optional argument, and a mandatory braced argument. The starred form expands to `\left\lvert #1 \right\rvert` where `#1` is the mandatory argument:

[7] See §3.1.11 for the `[p=.]` (which ensures the concluding full stop appears in the correct place.

`\eval[p=.]{\[ 3\abs*{\frac{\abs{n}}{21}-1} \]}[n=-7]` $\implies$

$$3\left|\frac{|n|}{21}-1\right|=2, \qquad (n=-7).$$

The optional argument provides access to the `\big...` modifiers:

```
\eval[p=.]{\[
  \abs[\Big]{\abs{a-c}-\abs[\big]{A-C}}
\]}[A=12,a=-10,C=7,c=-5]
```

$\implies$

$$\Big||a-c|-\big|A-C\big|\Big|=0, \qquad (A=12, a=-10, C=7, c=-5).$$

The form without either star or square bracket option dispenses with the modifiers altogether:

`\eval{$ \tfrac12(x+y)+\tfrac12\abs{x-y} $}[x=-3,y=7].` $\implies$
$\tfrac12(x+y)+\tfrac12|x-y|=7,\ \ (x=-3,y=7)$.

As noted, the star and square bracketed option are mutually exclusive arguments.

`numerica` also provides the functions `\floor` and `\ceil`, defined in the same way, taking a mutually exclusive star or square bracketed optional argument and for the starred forms expanding to `\left\lfloor #1 \right\rfloor` and `\left\lceil #1 \right\rceil` where `#1` is the mandatory argument, and for the square bracket option forms replacing the `\left` and `\right` with the corresponding `\big` commands. The form without star or square-bracket option dispenses with any modifier at all.

`\eval{$ \floor{-\pi} $}` $\implies \lfloor -\pi \rfloor = -4,$

`\eval{$ \ceil{\pi} $}` $\implies \lceil \pi \rceil = 4.$

The floor function, $\lfloor x \rfloor$, is the greatest integer $\leq x$; the ceiling function, $\lceil x \rceil$ is the smallest integer $\geq x$. Like the absolute value, the floor and ceiling functions, can be nested:

`\eval{$ \floor{-\pi+\ceil{e}} $}` $\implies \lfloor -\pi + \lceil e \rceil \rfloor = -1.$

2.4.11.1 Squaring, cubing, ... absolute values, etc.

These three functions can be raised to a power *without* extra parentheses:

`\eval{$ \ceil{e}^2 $},` $\implies \lceil e \rceil^2 = 9,$

`\eval{$ \abs{-4}^2 $}.` $\implies |-4|^2 = 16.$

2.4.12 Factorials, binomial coefficients

Factorials use the familiar trailing `!` notation:

`\eval{$ 7! $}` $\Longrightarrow 7! = 5040,$

`\eval{$ (\alpha+\beta)!-\alpha!-\beta! $}[\alpha=2,\beta=3]` $\Longrightarrow$
$(\alpha+\beta)!-\alpha!-\beta! = 112, \ \ (\alpha=2,\beta=3).$

The examples illustrate how `numerica` interprets the argument of the factorial symbol: it 'digests'

1. a preceding (possibly multi-digit) integer, or
2. a preceding variable token, or
3. a bracketed expression, or
4. a bracket-like expression – an absolute value, floor or ceiling function,

since they delimit arguments in a bracket-like way:

`\eval{$ \abs{-4}!+\floor{\pi}!+\ceil{e}! $}` $\Longrightarrow$
$|-4|!+\lfloor\pi\rfloor!+\lceil e\rceil! = 36.$

The result of feeding the factorial an expression different in kind from one of these four cases may give an error message or an unexpected result. Use parentheses around such an expression; for example write $(3^2)!$, rather than $3^2!$.

Nesting of brackets for factorials is accepted:

`\eval{$ ((5-2)!+1)! $}` $\Longrightarrow ((5-2)!+1)! = 5040.$

The factorials of negative integers or of non-integers are not defined in `numerica`. Again there is the problem met in relation to n-th roots of what happens if the argument of a factorial is the result of a calculation and rounding errors mean it is not an exact integer. This problem is unlikely to be of practical concern since `numerica` rounds the result of such a calculation by default to 14 significant figures before offering it to the factorial. Since `l3fp` works to 16 significant figures, there is ample 'elbowroom' to accommodate rounding errors before the result of a calculation ceases to round to an integer.

2.4.12.1 Double factorials

The double factorial, written $n!!$, is the product $n(n-2)(n-4)\ldots\times 4\times 2$ when n is even and the product $n(n-2)(n-4)\ldots\times 3\times 1$ when n is odd.

`\eval{$ 6!! $}` $\Longrightarrow 6!! = 48,$

`\eval{$ n!! $}[n=\sqrt{49}]` $\Longrightarrow n!! = 105, \ \ (n=\sqrt{49}),$

Since $n! = n!!(n-1)!!$ it follows that

$$n!! = \frac{n!}{(n-1)!!} = \frac{(n+1)!}{(n+1)!!}.$$

Putting $n = 0$ in the outer equality shows that $0!! = 1$. Now putting $n = 0$ in the left equality gives $(-1)!! = 1$. Double factorials therefore are defined for integers ≥ -1.

2.4.12.2 Binomial coefficients

Binomial coefficients are entered in LaTeX with the `\binom` command. It takes two arguments and has a text-style version `\tbinom` and a display-style version `\dbinom`. As implemented in `numerica`, these are *generalised* binomial coefficients:

$$\binom{x}{k} = \frac{x(x-1)\dots(x-k+1)}{k(k-1)\dots1}, \quad (x \in \mathbb{R},\ k \in \mathbb{N}),$$

where x need not be a non-negative integer, and where $\binom{x}{0} = 1$ by definition. Although the first (or upper) argument can be any real number, the lower argument *must* be a non-negative integer. Thus, `\eval{$ \tbinom53 $}` $\implies \binom{5}{3} = 10$, `\eval{$ \tbinom70 $}` $\implies \binom{7}{0} = 1$, `\eval{$ \tbinom{4.2}3 $}` $\implies \binom{4.2}{3} = 4.928$, but if the second (or lower) argument of `\binom` is *not* a non-negative integer, `numerica` displays a message; see §2.5.5.

2.4.13 Sums and products

`numerica` recognizes sums (`\sum` displaying as $\sum$) and products (`\prod` displaying as $\prod$), and expects both symbols to have lower and upper summation/product limits specified. The lower limit must be given in the form *sum/prod variable = initial value*; the upper limit requires only the final value to be specified (although it can also be given in the form *sum/prod variable = final value*). The values may be expressions depending on other variables and values but must evaluate to integers (or infinity – see §3.2). Evaluating to an integer means that they *round* to an integer, using a rounding value that is set by default to 14; (recall that `l3fp` works to 16 significant figures). If a limit evaluates to a non-integer at this 'int-ifying' rounding value, an error message results. (To change this 'int-ifying' rounding value, see §3.3.2.)

As an example of expressions in the limits, this example uses the floor and ceiling functions to convert combinations of constants to integers (the `[p]` is explained in §3.1.11),

`\eval[p]{\[ \sum_{n=\floor{\pi/e}}^{\ceil{\pi e}}n \]}` $\implies$

$$\sum_{n=\lfloor\pi/e\rfloor}^{\lceil\pi e\rceil} n = 45,$$

(which is $\sum_{n=1}^{9} n$). If the upper limit is less than the lower limit the result is zero. Notice that there is no vv-list. The summation variable does not need to be included there unless there are other variables that depend on it. However, in the case

`\eval[p]{\[ \sum_{k=1}^N\frac1{k^3} \]}[N=100][4]` $\Longrightarrow$

$$\sum_{k=1}^{N}\frac{1}{k^3}=1.202,\qquad (N=100),$$

the upper limit N is necessarily assigned a value in the vv-list.

To the author it seems natural to enter the lower limit first, immediately after the `\sum` command (the sum is *from* something *to* something), but no problem will accrue if the upper limit is placed first (after all, the appearance of the formula in the pdf is the same):

`\eval[p=.]{\[ \sum^N_{k=1}\frac1{k^3} \]}[N=100][4]` $\Longrightarrow$

$$\sum_{k=1}^{N}\frac{1}{k^3}=1.202,\qquad (N=100).$$

Another example of a sum, using binomial coefficients this time, is

`\eval[p]{\[ \sum_{m=0}^5\binom{5}{m}x^m y^{5-m} \]}[x=0.75,y=2.25]`
$\Longrightarrow$

$$\sum_{m=0}^{5}\binom{5}{m}x^m y^{5-m}=243,\qquad (x=0.75,y=2.25),$$

which is just `\eval{$(x+y)^5$}[x=0.75,y=2.25]` $\Longrightarrow$ $(x+y)^5=243$, $(x=0.75,y=2.25)$, or 3^5.

Now let's calculate a product:

```
\eval[p]{\[
  \prod_{k=1}^{100}
    \biggl(\frac{x^2}{k^2\pi^2} +1\biggr)
      \]}[x=1][3]
```

$\Longrightarrow$

$$\prod_{k=1}^{100}\biggl(\frac{x^2}{k^2\pi^2}+1\biggr)=1.174,\qquad (x=1),$$

to be compared with `\eval{$ \sinh 1 $}[3]` $\Longrightarrow$ $\sinh 1=1.175$. Obviously more terms than 100 are required in the product to achieve 3-figure accuracy.

2.4.13.1 Infinite sums and products

There is a strong urge to use ∞ in the upper limit of this product. Let's do so:

```
\eval[p=.]{\[
  \prod_{k=1}^{\infty}
    \biggl(\frac{x^2}{k^2\pi^2} +1\biggr)
      \]}[x=1][3]
```

$\Longrightarrow$

$$\prod_{k=1}^{\infty}\biggl(\frac{x^2}{k^2\pi^2}+1\biggr)=1.174,\qquad(x=1).$$

Disappointingly, we still get the same result, deficient by 1 in the third decimal place. Obviously `numerica` has not multiplied an infinite number of terms and, just as obviously, the finite number of terms it *has* multiplied are too few. How `numerica` decides when to stop evaluating additional terms in an infinite sum or product is discussed later, §3.2.

For this particular product the problem is that it converges slowly. Any criterion for when to stop multiplying terms or, for an infinite sum adding terms, seems bound to fail for some product or series. Presumably any stopping criterion must measure smallness in some way. But terms of the divergent harmonic series, $\sum(1/n)$ can always be found smaller than any value we care to specify. It is not surprising that a sufficiently slowly converging product or series falls foul of a given criterion.

The default criterion however can be changed. Because this involves values assigned in the settings option of the `\eval` command, I discuss infinite sums and products in the section discussing that optional argument; see §3.2.

Other infinite sums converge more rapidly, and the default settings work admirably. For example `\eval{$ (1+0.1234)^{4.321} $}` $\Longrightarrow (1{+}0.1234)^{4.321}=1.653329$. Using binomial coefficients we can express this as an infinite sum:

```
\eval[p=.]{\[
    \sum_{n=0}^{\infty}\binom{\alpha}{n}x^{n}
      \]}[\alpha=4.321,x=0.1234]
```

$\Longrightarrow$

$$\sum_{n=0}^{\infty}\binom{\alpha}{n}x^{n}=1.653329,\qquad(\alpha=4.321,x=0.1234).$$

2.5 Error messages

There are two kinds of error in `numerica`: those in the underlying LaTeX which are reported in the LaTeX log, shown on the terminal, and generally halt compilation, and specifically `numerica`-related errors which do not halt compilation and produce messages displayed in the pdf where one would expect the result of the calculation to be. The original reason for doing things this way was to enable `numerica` to be used effectively with the instant preview facility of the document processor LYX. More philosophically, one might view such errors as similar to grammatical errors or spelling mistakes in text. It is not clear that they should halt compilation. Hence strictly `numerica`-related errors leave brief messages in the pdf at the offending places.

Before discussing specific error messages, note that there is a debug facility (of a sort) discussed below in §3.1.1.

Error messages are in two parts: a *what* part and a *where* part.

2.5.1 Mismatched brackets

An unmatched left parenthesis or other left bracket (in this case a missing right parenthesis) usually results in a `numerica` error:

`$\eval{\sin(\pi/(1+x)}[x=1]$` $\Longrightarrow$!!! Unmatched `(` in: formula. !!!

For the same error in the vv-list, the what-part remains unchanged but the where-part is altered:

`$\eval{ 1+y }[x=1,y=\sin(\pi/(1+x)]$` $\Longrightarrow$
!!! Unmatched `(` in: variable = value list. !!!

The *what* message is the same; the *where* is different.

An unmatched right parenthesis or other right bracket (in this case a missing *left* parenthesis) usually results in a similar `numerica` error:

`$\eval{2((x+y)/(y+z)))^2}[x=1,y=2,z=3]$` $\Longrightarrow$
!!! Unmatched `)` in: formula. !!!

But note that an unmatched modifier like `\left` or `\right` is a LaTeX error and is caught by LaTeX before `numerica` can respond and so results in a terminal and logfile message.

2.5.2 Unknown tokens

An 'Unknown token' message can arise in a number of ways. If an expression involves a number of variables, some of which depend on others, their order in the vv-list matters:

`$\eval{\tfrac12 vt}[t=2,v=gt,g=9.8]$` $\Longrightarrow$
!!! Unknown token `t` in: variable = value list. !!!

The vv-list is evaluated from the *right* so that in this example the variable `v` depends on a quantity `t` that is not yet defined. Hence the message. The remedy is to move `t` to the right of `v` in the vv-list.

Similarly, if we use a variable in the formula that has not been assigned a value in the vv-list, we again get the 'Unknown token' message, but this time the location is the formula:

`$\eval{\pi r^2h}[r=3]$` $\Longrightarrow$!!! Unknown token `h` in: formula. !!!

The remedy obviously is to assign a value to `h` in the vv-list.

The same message will result if a mathematical operation or function is used that has not been implemented in `numerica`:

`$\eval{u \bmod v }[v=7,u=3]$` $\Longrightarrow$
!!! Unknown token `\bmod` in: formula. !!!

A missing comma in the vv-list will generally result in an unknown token message:

`$\eval{axy}[a=3 y=2,x=1]$` $\Longrightarrow$
!!! Unknown token `y` in: variable = value list. !!!

Because of the missing comma, `numerica` assumes `a` has the 'value' `3y=2` and has no knowledge of `y` as a variable.

The presence of multi-token variables can also cause this error message if the check for such variables is turned off; see §3.1.4.

2.5.3 Other vv-list errors

Other errors that can occur with the vv-list are overlooked value assignments to variables, or missing commas. For the first, it is essential that we do actually give a value to all variables occuring in the vv-list:

`$\eval{axy}[a=3,y=,x=1]$` $\Longrightarrow$!!! No value for y in: variable = value list. !!!

The remedy is obvious – assign a value to y.

Extra commas in the vv-list should cause no problems:

`$\eval{axy}[,a=3,,y=2,x=1,]$` $\Longrightarrow 6,\ \ (a = 3, y = 2, x = 1)$

2.5.4 Inverse powers of negative numbers

Inverse integer powers of positive numbers should always be possible, but raising a *negative* number to an inverse power generates an error even when, mathematically, it should not:

`\eval{$ (-125)^{1/3} $}` $\Longrightarrow$
!!! `l3fp` error 'Invalid operation' in: formula. !!!

This is a feature of floating point arithmetic. When a number is raised to a rational power, say p/q where p and q are non-zero integers, then the result is the p-th power of the q-th root of the number. Can a q-th root be taken? If our floating point system used (for ease of illustration) only 4 significant digits, $p/q = 1/3$ would be the fraction $3333/10^4$, an odd numerator over an even denominator. But a negative number does not possess an even (10^4-th) root.

2.5.5 Integer argument errors

Some functions require integer arguments – factorials, the second argument of a binomial coefficient, and (in `numerica`) n-th roots using the optional argument of `\sqrt`; also summation and product variables. If integers are explicitly entered for these arguments there is no problem, but if the value of the argument is the result of a calculation, rounding errors require thinking about. What accumulation of rounding errors is *too* much so that the result of the calculation *cannot* be considered an integer? `numerica` is generous: in the default setup, if a calculation rounds to an integer at rounding value 14 the result of the calculation is considered an integer (obviously, the integer resulting from the rounding). Since `l3fp` works to 16 significant figures that gives ample room for rounding errors to 'get lost in' and be ignored, while still ruling out such things as (recall the example in §2.3.4),

`\eval{\[ \sum_{n=1}^N n \]}[N=1/0.0123456789]` $\Longrightarrow$
!!! Integer required in: sum limits. !!!

where N differs from 81 not until the seventh decimal place.

The default rounding value of 14 for 'int-ifying' calculations can be changed: see §3.3.2.

2.5.6 Comparison errors

Should a user try to make a comparison using a combination like `>=` rather than `\geq`, `numerica` admonishes like this:

`$\eval{ \pi^e >= e^\pi }[?]$` $\Longrightarrow$
!!! Multi-token comparison in: formula. !!!

(The relation is false by the way.)

2.5.7 Invalid base for `\log`

ISO recommends using `\log` only with a subscripted base specified. Otherwise how is one to know whether the base is e or 10 or 2 or whatever? Nonetheless `numerica` assumes that when `\log` is used unsubscripted, the base is 10. Suppose you want to make 12 the base, but forget to put braces around the 12:

`$\eval{ \log_12 1728 }$` $\Longrightarrow$
!!! Valid base required for `\log` in: formula. !!!

Here, `numerica` has taken `1` as the base (and 21728 as the argument) of the logarithm and responds accordingly.

2.5.8 `l3fp` errors

Some errors arising at the `l3fp` level are trapped and a message displayed.

- Dividing by zero

`$\eval{1/\sin x}[x=0]$` $\Longrightarrow$!!! `l3fp` error 'Division by zero' in: formula. !!!

Note however that `$\eval{1/\sin x}[x=\pi]$` $\Longrightarrow$ 4193528956200936, $(x = \pi)$, because of rounding errors in distant decimal places. No doubt this is true for other functions as well.

- Invalid operation

`$\eval{\arccos x}[x=2]$` $\Longrightarrow$
!!! `l3fp` error 'Invalid operation' in: formula. !!!

In this case the inverse cosine has been fed a value of x outside its domain of definition, the interval $[-1, 1]$. Trying to evaluate an expression that resolves to $0/0$ also produces this message:

`$\eval{\frac{1-y}{x-2}}[x=2,y=1]$`$\Longrightarrow$
!!! `l3fp` error 'Invalid operation' in: formula. !!!

- Overflow/underflow

The factorial (discussed in §2.4.12) provides an example of overflow:

`$\eval{3249!}$` $\Longrightarrow$!!! `l3fp` error 'Overflow' in: formula. !!!

This is hardly surprising since

`$\eval{3248!}[x]$` $\Longrightarrow$ $1.973634 \times 10^{9997}$.

There is a limit on the size of exponents that `l3fp` can handle. A number in the form $a \times 10^b$ must have $-10001 \le b < 10000$. If this is not the case an overflow or underflow condition occurs. As the examples show, an overflow condition generates a `numerica` error. For underflow, where the number is closer to 0 than 10^{-10001}, `l3fp` assigns a zero value to the quantity. `numerica` accepts the zero value.

Chapter 3

Settings

A calculation is effected against a background of default values for various quantities. For a particular calculation, these values may not be appropriate; or you may have different preferences. The way to change settings for a particular calculation is through the settings option of `\nmcEvaluate` discussed next. The way to change a *default* setting is by creating a configuration file `numerica.cfg` discussed in §3.3.

3.1 Settings option

The second argument of the `\nmcEvaluate` command is the settings option, delimited by square brackets. This option is a *key=value* list, hence comma-separated. *Key=value* lists tend to be wordy. For back-of-envelope calculations one wants to be able to 'dash off' the calculation, hence the short, cryptic nsture of the keys. Most settings are generic, applicable not only to `\nmcEvaluate` but also to other commands that are available if `numerica` is loaded with the `plus` option; see the associated document `numerica-plus.pdf`.

3.1.1 'Debug' facility

It is rather grandiose to call this a debug facility, but if a calculation goes wrong or produces a surprising result, `numerica` offers a means of examining some intermediate stages on the way to the final result. To use the facility, enter

```
dbg = <integer>
```

into the settings option. (White space around the equals sign is optional.)

- `dbg=0` turns off the debug function, displays the result or error message (this is the default);
- `dbg=1` equivalent to `dbg=2*3*5*7`;

Table 3.1: Settings options

key	type	meaning	default
`dbg`	int	debug 'magic' integer	`0`
`reuse`	int	form of result saved with `\nmcReuse`	`0`
`^`	char	exponent mark for sci. notation input	`e`
`xx`	int (0/1)	multi-token variable switch	`1`
`()`	int (0/1/2)	trig. arg. parsing	`0`
`o`		degree switch for trig. funcions	
`log`	num	base of logarithms for `\log`	`10`
`vvmode`	int (0/1)	vv-list calculation mode	`0`
`vvd`	token(s)	vv-list display-style spec.	`{,}\mskip 12mu plus 6mu minus 9mu(vv)`
`vvi`	token(s)	vv-list text-style spec.	`{,}\mskip 36mu minus 24mu(vv)`
`*`		suppress equation numbering if `\\` in `vvd`	
`p`	token(s)	punctuation (esp. in display-style)	`,`

The 'magic' integers are the following primes and their products:

- `dbg=2` displays the vv-list after multi-token variables have been converted to their single token form, `\_nmca`, `\_nmcb`, etc.;
- `dbg=3` displays the formula after multi-token variables have been converted to their single token form;
- `dbg=5` displays the stored variables and their evaluated values (`dbg=2` lists the values as expressions; here they have been evaluated); note that any saved values (Chapter 6) that have been loaded will also feature in this list which might lead to a messy display depending on the nature of those values;
- `dbg=7` displays the formula after it has been fp-ified (but before it has been fed to `l3fp` to evaluate);
  - should the formula successfully evaluate, the result of the evaluation is also displayed.

To display two or more of these elements simultaneously, use the product of their debug numbers for the magic integer. This can be entered either as the multiplied-out product, or as the 'waiting to be evaluated' product with asterisks (stars) between the factors. Thus `dbg=6` or `dbg=2*3` display both the vv-list and formula after multi-token variables have been converted to single token form; `dbg=10` or `dbg=2*5` display both the vv-list after multi-token variables have been converted to single token form and the recorded variables with their evaluated values. And similarly for the other magic integers listed. For other integers, if they are divisible by 2 or 3 or 5 or 7, they will display the corresponding component. Both `dbg=210` and `dbg=2*3*5*7` display all four elements, but rather than remembering this product, it suffices to put `dbg=1`. This is equivalent and displays all elements.

Table 3.2: Magic integers

integer	factors
6	2,3
10	2,5
14	2,7
15	3,5
21	3,7
30	2,3,5
35	5,7
42	2,3,7
70	2,5,7
105	3,5,7
210	2,3,5,7

The debug option uses an `aligned` or `align*` environment to display its wares, depending on the presence or absence of math delimiters around the `\eval` command. In the next example I have used multi-token variables to illustrate the different elements in the debug display, and a chain of comparisons to show how `numerica` treats these (§2.3.4).

```
\eval[dbg=1]{ a_1<2a_2<3a_3<\pi+e }
  [a_1=\pi,a_2=\phi,a_3=e\gamma][6?'']
```

$\Longrightarrow$

```
  vv-list:  \_nmcg =e\gamma , \_nmcf =\phi , \_nmce =\pi
 formula:  \_nmce <2\_nmcf <3\_nmcg <\pi +e
  stored:  \_nmcg =1.569034853003742, \_nmcf =1.618033988749895, \_nmce
           =3.141592653589793
 fp-form:  round((3.141592653589793)-
           (2(1.618033988749895)),6)<0&&round(2(1.618033988749895)-
           (3(1.569034853003742)),6)<0&&round(3(1.569034853003742)-
           ((pi)+exp(1)),6)<0
  result:  T
```

Note that the four elements are displayed in temporal order: first comes the vv-list after conversion of multi-token to single-token variables, then the formula in the single-token variables; these are created essentially at the same time. The vv-list is presented in left-to-right order because that is the direction of evaluation *internally* in `numerica`. Next the stored values of the variables are displayed. These are the values *after* evaluation. The fourth element both in the display and chronologically is the fp-ified formula; this is often a thicket of parentheses. The final element of the display and chronologically is the result of evaluating the formula. This is displayed only if 7 is a factor of the `dbg` integer, and there is no error.

When interpreting the fp-form, one should be aware of differences between `numerica` and `l3fp`. In particular be aware that in `l3fp` function calls bind most tightly so that, for example, `sin 2pi` evaluates not to zero but to $(\sin 2) \times \pi$ and `sin x^2` evaluates to $(\sin x)^2$. This should not be of any concern to the user except as here in debug mode when interpreting fp-forms.

Finally, note that those mathematical operations that have no direct representation in `l3fp` contribute only their value to the fp-form. This applies to sums and products, double factorials and partly to binomial coefficients as illustrated in the followng (ridiculous) example:

```
\eval[dbg=1]{\[
  \sum_{n=1}^5 n + \binom{10}{m}
    + \prod_{n=2}^5 (1-1/n) + m!! \][m=6]
```

$\Longrightarrow$

```
 vv-list:  m=6
formula:  \sum _{n=1}^5 n+\binom {10}{m} +\prod _{n=2}^5(1-1/n) +m!!
  stored:  m=6
fp-form:  15+(151200/720)+0.2+(48)
   result:  273.2
```

The various contributions to the overall result are displayed simply as numbers because `l3fp` does not (at least as yet) handle these elements natively.

3.1.1.1 Negative `dbg` values

Negative `dbg` values are possible: `dbg=-2`, `dbg=-3`, etc. (and `dbg=-1` meaning `dbg=-210`) have exactly the same effects as the corresponding positive values except for some details of display. The display for positive `dbg` values is the one evident in the examples above. Lines wrap, the left margin is not indented and the display occupies the page width. For negative `dbg` values, lines do not wrap, the left margin is indented and the display occupies the text width. An example is presented in §4.2.1 below where the display for a nested `\eval` is significantly improved with a negative `dbg` value.

3.1.2 Reuse setting

This setting determines whether the entire display or only the numerical result is saved to file with the `\nmcReuse` command. See below, Chapter 6, §6.3.2.

3.1.3 Inputting numbers in scientific notation

Outputting numbers in scientific notation is controlled by the final trailing argument of the `\eval` command. That is turned off by default and needs to be explicitly ordered. Similarly, *inputting* numbers in scientific notation is turned off by default and needs to be explicitly ordered. To turn it on, write

```
    ^ = <char>
```

in the settings option, where `<char>` is any single character, usually `e` or `d` or their upper-casings, but not restricted to them: `^=@` for instance is perfectly possible, and has the advantage over `e` or `d` that it doesn't conflict with the use of the character as a variable or constant.

`$ \eval[^=@]{ 1.23@-1 } $` $\Longrightarrow$ 0.123.

With letters for the exponent mark – say `d` or `e` – the problem is interpreting forms like `8d-3` or `2e-1`. Does such a form denote a number in scientific notation or an algebraic expression? In `numerica`, if the settings option shows `^=d`, then a form like `8d-3` is treated as a number in scientific notation. Similarly for `e` or any other letter used as the exponent marker for the input of scientific numbers. (But only one character can be so used at a time.) Note that the number *must* start with a digit: `e-1` for instance does not and will be treated as an algebraic expression involving the exponential constant (unless `e` is assigned a different value in the vv-list).

`$ \eval[^=e]{ x+e-1 }[x=2e-1] $` $\Longrightarrow 1.918282, \quad (x = 2e - 1).$

The problem here is that `2e-1` is treated as a number in scientific notation but displays in the vv-list as if it were an algebraic expression. The solution is to put `2e-1` into an `\mbox` or `\text` command in the vv-list:

`$ \eval[^=e]{ x+e-1 }[x=\text{2e-1}] $` $\Longrightarrow 1.918282, \quad (x = \text{2e-1}).$

If you use a particular character as the exponent marker for inputting numbers in scientific notation, it is good practice *not* to use that character as a variable, not because it will cause an error but because it makes expressions harder to read.

3.1.4 Multi-token variables

Variables need not consist of a single character or token (like x or α). Multi-token symbols like x' or t_i or var are perfectly acceptable. For its internal operations, `numerica` converts such multi-token names to single tokens (as discussed in §2.2.1). This conversion takes time. Even if there are no multi-token variables used at all, `numerica` still needs to check that that is so. There is a setting that allows a user to turn off or turn on the check for such variables by entering

```
    xx = <integer>
```

into the settings option. If `<integer>` is `0`, the check for (and conversion of) multi-token variables is turned off; if `<integer>` is `1` (or any other *non-zero* integer), the check, and conversion if needed, goes ahead. By default, checking for multi-token variables and converting them if found is turned *on*. (The name for the key, `xx`, is chosen because `x` is the most familiar variable of all, introduced in elementary algebra, and doubling it like this suggests multi-token-ness.)

If checking is turned off when a multi-token variable is present, an error results. We don't need to enter `xx=1` in the first of the following examples because the check for multi-token variables is on by default. Explicitly turning it off in the second produces an error.

`\eval{$ x_0^{\,2} $}[x_0=5]` $\implies x_0^{\,2} = 25, \quad (x_0 = 5),$

`\eval[xx=0]{$ x_0^{\,2} $}[x_0=5]` $\implies$
!!! Unknown token `x` in: formula. !!!

3.1.5 Parsing arguments of trigonometric functions

This setting allows a wider range of arguments to trigonometric functions to be used (think Fourier series) without needing to insert extra parentheses in order for them to be read correctly by `\eval`; see §3.4.2.3.

3.1.6 Using degrees rather than radians

You may find it more convenient to use degrees rather than radians with trigonometric functions.This can be switched on simply by entering a lowercase `o` in the settings option. (The author's fond hope is that the charitable eye might see a degree symbol in the `o`.) Thus

`\eval[o]{$ \sin 30 $}` $\implies \sin 30 = 0.5,$
`\eval[o]{$ \arcsin 0.5 $}` $\implies \arcsin 0.5 = 30.$

3.1.7 Specifying a logarithm base

If you wish to use `\log` without a subscripted base in a particular calculation, then add an entry like

```
    log = <positive number≠1>
```

to the settings option of the `\eval` command. The `<positive number>` does not need to be an integer. It could be `e` (if you object to writing `\ln`) but is more likely to be 2 or another small integer; 10 is the default. If you want to use this changed base not for one but most calculations, then add an entry with your choice of base to a configuration file; see §3.3.

3.1.8 Calculation mode

A variable may change in the course of a calculation. This is certainly true of sums and products. If a parameter in the vv-list depends on the variable then that parameter will need to be recalculated, perhaps repeatedly, in the course of a calculation. By entering

```
vvmode = <integer>
```

in the settings option it is possible to turn on or off the ability to repeatedly evaluate the vv-list; `<integer>` here takes two possible values, `0` or `1`. `vvmode=0` means the vv-list is evaluated once at the start of the calculation; `vvmode=1` means the vv-list is recalculated every time the relevant variable changes.

For example, it may be desirable to place the summand, or some part of it, in the vv-list. Since the summation variable obviously changes during the course of the calculation, we need to enter `vvmode=1` in the settings option. Repeating an earlier sum (the seting `p=.` is discussed in §3.1.11)

```
\eval[p=.,vvmode=1]{\[ \sum_{k=1}^N f(k) \]}
  [N=100,f(k)=1/k^3,{k}=1][4]
```

$\Longrightarrow$

$$\sum_{k=1}^{N} f(k) = 1.202, \qquad (N = 100, f(k) = 1/k^3).$$

As you can see, the summand `f(k)` has been given explicit form in the vv-list – equated to `1/k^3`. That means we need to give a preceding value to `k` in the vv-list; hence the rightmost entry. But we don't want `k=1` appearing in the final display, so we wrap `k` in braces (see §2.2.3.2). Since the value `k=1` applies only to the first term in the sum, to ensure it is not used for all terms, we enter `vvmode=1` in the settings option. This turns vv-recalculation mode on and ensures `k=1` is overwritten by `k=2`, `k=3` and so on, and the vv-list recalculated each time. The final result is the same as before, although recalculating the vv-list at each step is a more resource-hungry process. The difference may not be marked for this example; with more complicated expressions it noticeably takes longer.

Because it is necessary to activate this switch when using *implicit* notations – like $f(k)$ in the example – rather than the explicit form of the function in the main argumet it seems natural to call `vvmode=1` *implicit* mode and `vvmode=0` (the default) *explicit* mode. Most calculations are explicit mode – the vv-list is evaluated only once.

3.1.9 Changing the vv-list display format

In previous formulas with variables the vv-list has been displayed following the result. It is wrapped in parentheses following a comma followed by a space. These formatting elements – comma, space, parentheses – can all be changed with the settings option.

The default format specification is

```
{,}\mskip 12mu plus 6mu minus 9mu(vv)
```

for a text-style display (an inline formula) and

```
{,}\mskip 36mu minus 24mu(vv)
```

in a display-style context. The commas are wrapped in braces because these are items in a comma-separated list. Both entries exhibit the elements: punctuation (comma), preceding a variable space, preceding the parenthesized vv-list (the `vv` placeholder). No full stop is inserted after the closing parentheses because the `\eval` command may occur in the middle of a sentence (even in display style). For inline use, the elasticity of the space becomes relevant when TEX is adjusting individual lines to fit sentences into paragraphs and paragraphs into pages. The largest spacing that can be stretched to is a quad, 18 mu (mu = math unit), and the smallest that can be shrunk to is a thin space, 3 mu. In display style, the largest spacing specified is the double quad, in line with the recommendation in *The TEX Book*, Chapter 18, but this can shrink to a single quad, for instance if the vv-list is heavily populated with variables so that the evaluated result is pushed well to the left by the vv-list. (But see below, §3.1.10.)

If you want to change these defaults, enter in the settings option

```
vvi = <new specification>
```

to change the inline display and

```
vvd = <new specification>
```

to change the display-style display For example the settings

```
vvi = {,}\quad(vv)
vvd = {,}\qquad(vv)
```

would give a comma (in braces since the settings option is a comma-separated list) and a fixed space (of one or two quads) between the result and the parenthesized vv-list.

The vv-list itself in the display specification is represented by the placeholder `vv`. If the `vv` is omitted from the specification, then the vv-list will not appear at all:

`\eval[vvi=?!]{$ \pi $}[\pi=3]` $\Longrightarrow \pi = 3?!$

More relevantly, it may well be the case that all variables in the vv-list are suppressed (wrapped in braces). In that case the display would look something like `, ()`. To prevent this enter `vvi=` in the vv-list, in the inline case, or `vvd=` in the display-style case, i.e. enter an empty value. (Alternatively, use the star option of the `\eval` command.)

Another minor wrinkle occurs if you want to change parentheses around the vv-list to square brackets. Because the settings option is a square-bracket delimited argument, the square brackets in the specification will, like commas, need to be hidden in braces, although you can get away with braces around the whole spec.:

```
vvi={,\mskip 12mu plus 6mu minus 9mu [vv]}
```

3.1.10 Displaying the vv-list on a new line

Display of a long formula with many variables, hence a full vv-list, may not fit comfortably on a line. In an earlier example I used Brahmagupta's formula to calculate the area of a triangle. It squeezed onto a line. I shall now use his formula for the area of a cyclic quadrilateral:

$$A = \sqrt{(s-a)(s-b)(s-c)(s-d)}.$$

The extra side (quadrilateral as against triangle) means there is a further variable to accommodate, not only in the formula but also in the vv-list. In the following example, the cyclic quadrilateral is formed by a 45-45-90 triangle of hypotenuse 2 joined along the hypotenuse to a 30-60-90 triangle. The sides are therefore $\surd2, \surd2, \surd3, 1$. Adding the areas of the two triangles, the area of the quadrilateral is $A = 1 + \frac{1}{2}\surd3$, or in decimal form, `$\eval{1+\tfrac12\surd3}$` $\Longrightarrow$ 1.866025. Let's check with Brahmagupta's formula:

```
\eval[p=.,vvd={,}\\(vv),*]
  {\[ \sqrt{(s-a)(s-b)(s-c)(s-d)} \]}
    [s=\tfrac12(a+b+c+d),
       a=\surd2,b=\surd2,c=\surd3,d=1]
```

$\Longrightarrow$

$$\sqrt{(s-a)(s-b)(s-c)(s-d)} = 1.866025,$$
$$(s = \tfrac{1}{2}(a+b+c+d), a = \surd2, b = \surd2, c = \surd3, d = 1).$$

The values agree. The point to note here is the `vvd={,}\\(vv)` and the `*` in the settings option. The `\\` in a specification for `vvd` acts as a trigger for `numerica` to replace whatever math delimiters are enclosed by the `\eval` command with a `multline` environment. As you can see, the specification inserts a comma after the formula and places the parenthesized vv-list on a new line. The star `*` if present suppresses equation numbering by turning the `multline` into a `multline*` environment.

Things to note in the use of `\\` in a `vvd` specification are that

- it applies only to the `vvd` specification, not the `vvi` spec.;
- it applies only when `\eval` *wraps around* a math environment of some kind;
- it has no effect when the `\eval` command is used *within* a math environment when the presentation of the result is of the form *result, vv-list*. The formula is not displayed and so the pressure on space is less and the 'ordinary' vv-list specification is used.

3.1.11 Punctuation

The `\eval` command can be used within mathematical delimiters or it can be wrapped around mathematical delimiters. The latter gives a *formula=result* style of display automatically, which is convenient. One doesn't need to write the *formula=* part of the expression, but it causes a problem when `\eval` wraps around a display-style or similar environment: how to display a following punctuation mark? For an inline display we can simply follow the `\eval` command with the appropriate punctuation, for instance: `\eval{$ 1+1 $}.` $\Longrightarrow 1+1=2$. But with `\[ \]` delimiters used *within* the `\eval` command – `\eval{\[ 1+1 \]}.` – the fullstop slides off to the start of the next line, since it is beyond the closing delimiter. We want it to display as if it were the last element *before* the closing delimiter.

Explicitly putting it there – `\eval{\[ 1+1. \]}` – means the punctuation mark becomes part of the formula. Potentially `numerica` then needs to check not just for a fullstop but also other possible punctuation marks like comma, semicolon, perhaps even exclamation and question marks. All these marks have roles in mathematics or `l3fp`. Including them in the formula means distinguishing their punctuation role from their mathematical role and can only cause difficulties (and code bloat).

Instead, `numerica` uses the setting

```
p = <char(s)>
```

to place the `char(s)` after the result but within the environment delimiters. The default punctuation mark is the comma so that simply entering `p` will produce a comma in the appropriate place. This saves having to write `p={,}` as would otherwise be required, since the settings option is a *comma*-separated list.

Nor is one limited to a single punctuation mark:

`\eval[p=\ (but no 8!)]{\[ \frac{1}{81} \]}[9]` $\Longrightarrow$

$$\frac{1}{81} = 0.012345679 \text{ (but no 8!)}$$

Table 3.3: Settings for infinite sums & products

key	type	meaning	default
`S+`	int	extra rounding for stopping criterion	`2`
`S?`	int ≥ 0	stopping criterion query terms for sums	`0`
`P+`	int	extra rounding for stopping criterion	`2`
`P?`	int ≥ 0	stopping criterion query terms for products	`0`

3.2 Infinite sums and products

There are ways of tweaking various default settings to nudge infinite sums and products to a correct limit. These tweaks are applied via the settings option of the `\eval` command.

The normal convergence criterion used by `numerica` to determine when to stop adding/multiplying terms in an infinite sum/product is *when the next term added/multiplied leaves the total unaltered when rounded to 2 more digits than the specified rounding value.* Suppose T_k is the sum/product after the inclusion of k terms, and r is the rounding value. Denote T_k rounded to r figures by $(T_k)_r$. *The infinite sum or product stops at the* $(k+1)$*-th term (and the value is attained at the* k*-th term) when* $(T_{k+1})_{r+2} = (T_k)_{r+2}$. The hope is that if this is true at rounding value $r+2$ then at rounding value r the series or product will have attained a stable value at that level of rounding.

For a series of monotonic terms converging quickly to a limit, this stopping criterion works well, less so if convergence is slower, as seen earlier with the infinite product for $\sinh 1$. The criterion can fail completely when terms behave in a non-monotonic manner. Distant terms of a Fourier series, for example, may take zero values; the criterion is necessarily satisfied but the series may still be far from its limit. In a product the equivalent would be a distant term taking unit value. Such series or products may also have initial 'irregular' terms including zero/unit terms. A summation/product might stop after only one or two additions/multiplications if the criterion were applied to them.

To cope with these possibilities, `numerica` offers two settings for sums, two for products, summarized in Table 3.3. These are entered in the settings option of the `\eval` command.

- `S+=<integer>` or `P+=<integer>` additional rounding on top of the specified (or default) rounding for the calculation; default = 2
  - the larger the additional `<integer>` is, the more likely that sum or product has attained a stable value at the specified rounding r

- `S?=<integer` $\geq$ `0>` or `P?=<integer` $\geq$ `0>` the number of final terms to query after the stopping criterion has been achieved to confirm that it is not an 'accident' of particular values; default = 0
  - a final few terms to be summed/multiplied and the rounded result after each such operation to be compared with the rounded result at the time the stopping criterion was achieved. Suppose the additional rounding (`S+` or `P+`) is n on top of the specified rounding r and let the number of final checking terms be m. Suppose T_{k_0} is the first term at which the stopping criterion is achieved: $(T_{k_0})_{r+n} = (T_{k_0+1})_{r+n}$. What we require of the final query terms is that $(T_{k_0})_{r+n} = (T_{k_0+1+j})_{r+n}$ for $j = 0, 1, \ldots, m$.

Previously we found that the infinite product for sinh 1 with the default settings gave the wrong value, 0.174, deficient by 1 in the last digit. We now have the means to tweak the stopping criterion by increasing the additional rounding:

```
\eval[p,P+=3]{\[
  \prod_{k=1}^{\infty}
  \biggl(\frac{x^2}{k^2\pi^2} +1\biggr)
\]}[x=1][3] \nmcInfo{prod}.
```

$\Longrightarrow$

$$\prod_{k=1}^{\infty}\biggl(\frac{x^2}{k^2\pi^2}+1\biggr) = 1.175, \qquad (x=1),$$

350 factors.

To obtain that last item of information (350 factors), I've anticipated a little and used the command `\nmcInfo` with the argument `prod`; see Chapter 5. The product now produces the correct three-figure value, but it takes 350 factors to do so.

Knowing how many terms or factors have been needed helps assess how trustworthy the result from an infinite sum or product is. For example, for the exponential series,

```
\eval[p]{\[
  \sum_{k=0}^\infty \frac1{k!}
\]}[9] \nmcInfo{sum}.
```

$\Longrightarrow$

$$\sum_{k=0}^{\infty}\frac{1}{k!} = 2.718281828,$$

15 terms.

To 9 places of decimals, using the default value `S+=2`, the exponential series arrives at the right sum after only 15 terms. Convergence is rapid. We can trust this result (and it is in fact the correct nine-figure value). By contrast, if we didn't know the value of sinh 1 beforehand, noting the number of factors required would make us justly cautious about accepting the result of the infinite product calculation.

One way to gain confidence in a result is to choose a possibly unrealistic rounding value – say, the default 6 for the infinite product then use *negative* values for the extra rounding, `S+=-5`, `S+=-4`, . . . , so that the stopping criterion applies at rounding values s of $6 + (-5) = 1$, $6 + (-4) = 2$, and so on, but the result is always presented to 6 decimal places. One can then see how the 6-figure results behave relative to the number of terms it takes to meet the stopping criterion. A little experimenting shows that for our infinite product for $\sinh 1$ the number of factors N_s at a stopping rounding value s increases in geometric proportion with a scale factor of about 3: $N_{s+1}/N_s \approx 3$. For the exponential series on the other hand $N_s = 4+s$, the number of terms increasing in direct proportion to the stopping rounding value.

A similar calculation for the sum of inverse fourth powers of the integers $\zeta(4) = \sum_{n=1}^{\infty} \frac{1}{n^4}$, inverse third powers, $\zeta(3)$, and inverse squares, $\zeta(2)$, using `\nmcInfo` to find how many terms are required at each stopping rounding value, shows that at least over the rounding value range 1 to 8, for inverse fourth powers $N_{s+1}/N_s \approx 1.7$, for inverse third powers $N_{s+1}/N_s \gtrsim 2$ and for inverse squares $N_{s+1}/N_s \approx 3$. All are geometric rather than arithmetic progressions, but for inverse fourth powers the scale factor (≈ 1.7) is sufficiently small that for these low values of s the number of terms required does not grow too quickly. It is a standard result (Euler) that the series sums to $\pi^4/90$: `$ \eval{ \pi^4/90 } $` $\Longrightarrow$ 1.082323 to six places, and indeed, with the default `S+=2`,

`\eval[p]{\[ \sum_{k=1}^\infty \frac1{k^4} \]}` $\Longrightarrow$

$$\sum_{k=1}^{\infty} \frac{1}{k^4} = 1.082323,$$

there is complete agreement.

For inverse third powers, the number of terms required to reach the stopping criterion grows rapidly for rounding values from 7 onwards ($2^7 = 128$, $2^8 = 256$, . . .). This suggests trying for a five-figure result (with the default setting `S+=2` the stopping rounding value is 7). Doing this gives a result 1.20205 to five decimal places. *HMF* Table 23.3 has this quantity tabulated to 20 places and shows our result is too small by 1 in the final figure.

For inverse second powers, the number of terms required to reach the stopping criterion increase even more quickly: $3^4 = 81$, $3^5 = 243$, and so on. A three figure answer (with the default setting `S+=2` the stopping rounding value is 5) seems the best we can hope for. Doing the evaluation gives 1.642 whereas we know that $\zeta(2) = \pi^2/6$ (Euler's famous result), evaluating to 1.645. Even with `S+=3`, the sum is still too small, 1.644 after 1007 terms. Increasing the additional rounding to 4, `S+=4`, does finally give the correct three-figure result, 1.645, but only after summing 3180 terms.

3.2.1 Premature ending of infinite sums

All the series considered so far have been monotonic. Trigonometric series will generally not be so, nor even single-signed.

Trigonometric sums are computationally intensive and so, for the following example, I have specified a rounding value of 2. The series

$$\sum_{n=1}^{\infty}\frac{4}{n^2\pi^2}(1-\cos n\pi)\cos 2\pi nt$$

is the Fourier series for the triangular wave function $\wedge\!\wedge\!\wedge\!\wedge\ \ldots$ of period 1, symmetric about the origin where it takes its maximum value 1, crossing the axis at $t = 0.25$ and descending to its minimum -1 at $t = 0.5$, before ascending to a second maximum at $t = 1$ (and so on). In the interval $[0, 0.5)$ the series should sum to $1-4t$. The problem is that the summand $\frac{4}{n^2\pi^2}(1-\cos n\pi)\cos 2\pi nt$ vanishes both when n is even and when $4nt$ is an odd integer. If $t = 0.1$ then $4nt$ is never an odd integer so the summand vanishes only for n even, every second term. We expect the result to be $1 - 4 \times 0.1 = 0.6$.

```
\eval[p]{\[
  \sum_{n=1}^{\infty}
    \frac{4}{n^{2}\pi^{2}}
    (1-\cos n\pi)\cos2\pi nt
\]}[t=0.1][2] \nmcInfo{sum}.
```

$\Longrightarrow$

$$\sum_{n=1}^{\infty}\frac{4}{n^2\pi^2}(1-\cos n\pi)\cos 2\pi nt = 0.66, \qquad (t = 0.1),$$

1 term.

Only one term? Of course – since the second term n is even; the term vanishes and the stopping criterion is satisfied. The way around this problem is to query terms *beyond* the one where the stopping criterion is achieved, i.e., to set `S?` to a nonzero value. We try `S?=1`:

```
\eval[p,S?=1]{\[
  \sum_{n=1}^{\infty}
    \frac{4}{n^{2}\pi^{2}}
    (1-\cos n\pi)\cos2\pi nt
\]}[t=0.1][2] \nmcInfo{sum}.
```

$\Longrightarrow$

$$\sum_{n=1}^{\infty}\frac{4}{n^2\pi^2}(1-\cos n\pi)\cos 2\pi nt = 0.6, \qquad (t = 0.1),$$

65 terms.

Table 3.4 lists the results of evaluating the *finite* sums from $n = 1$ to N for values of N around 65. Since we have specified a rounding value of 2 for the calculation, the stopping criterion applies at a rounding value of 2 more than that, 4. Since $N = 64$ is even, the summand for the 64th term is zero and the sum takes the same value as for $N = 63$. The 65th term is the query term and the sum differs, so the summation continues. The 66th term vanishes, so the stopping criterion is met. This time for the query term, the 67th, the sum retains the same 4-figure value, and the summation stops. The result was attained at the 65th term.

Table 3.4: Finite sums

N	Σ
63	0.6001
64	0.6001
65	0.5999
66	0.5999
67	0.5999

Should we be confident in the result? Increase the number of query terms to 3 (there is no point in increasing `S?` to 2 because of the vanishing of the even terms), the sum stops after 113 terms, with the same 0.6 result. Indeed, increasing `S?` to $5, 7, \ldots$ makes no difference. It still takes 113 terms to reach the stable two-figure result 0.6.

For a final example, consider the error function

$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

which can also be rendered as an infinite sum (*HMF* 7.1.5):

$$\operatorname{erf} z = \sum_{n=0}^{\infty} (-1)^n \frac{z^{2n+1}}{n!(2n+1)}.$$

(`\erf` expanding to `erf` has been defined in the preamble to this document using `\DeclareMathOperator`.) We calculate this sum for $z = 2$ to 10 places of decimals. Although this is an alternating series, it is obvious that the summand never vanishes when $z \neq 0$ as here. Hence there seems no need to change the default value `S?=0`.

```
\eval[p]{\[
  \frac2{\sqrt{\pi}}
    \sum_{n=0}^\infty(-1)^n
      \frac{z^{2n+1}}{n!(2n+1)}
\]}[z=2][10*] \nmcInfo{sum}.
```

$\Longrightarrow$

$$\frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} (-1)^n \frac{z^{2n+1}}{n!(2n+1)} = 0.9953222650, \qquad (z = 2),$$

26 terms.

According to *HMF* Table 7.1, this calculated value of erf 2 is correct to all 10 places. But beyond $z = 2$ errors will begin to interfere with the result. Note that 26 terms means $n = 26$ was the last value of n for which the summand was evaluated. (The sum stops at the 26th term, $n = 25$, but the next term $n = 26$

needs to be calculated for the stopping criterion.) Fortuitously, $2^{2\times26+1} = 2^{53}$ is the greatest power of 2 that can be *exactly* rendered to the 16 significant figures that `l3fp` uses. But $n!$ exceeds the 16-significant figure limit of `l3fp` when $n > 21$, so despite the 10-figure result, errors have already begun to occur in the denominator of the summand and accrue in the sum when $z = 2$. For larger z values the errors can only get worse and at some point will render the calculated value worthless at any meaningful rounding value. For example, when $z = 7$ the sum apparently 'evaluates' to over 929 whereas we know that

$$\operatorname{erf} z < \frac{2}{\sqrt{\pi}} \int_0^\infty e^{-t^2} dt = 1.$$

3.2.2 Double sums or products

Sums or products can be iterated. For instance, the exponential function can be calculated this way:

```
\eval[p]{\[ \sum_{k=0}^{\infty}\prod_{m=1}^{k}\frac{x}{m} \]}[x=2]
```

$$\Longrightarrow$$

$$\sum_{k=0}^{\infty}\prod_{m=1}^{k}\frac{x}{m} = 7.389056, \qquad (x = 2),$$

which is `\eval{$ e^2 $}` $\Longrightarrow$ 7.389056.

A second example is afforded by Euler's transformation of series (*HMF* 3.6.27). To calculate e^{-1} we use

```
    \eval[p={,}\quad \mbox{\nmcInfo{sum}}.]
      {\[ \sum_{n=0}^{\infty}\frac{(-1)^{n}}{n!} \]}[3]
```

$\Longrightarrow$

$$\sum_{n=0}^{\infty}\frac{(-1)^n}{n!} = 0.368, \quad 9 \text{ terms.}$$

(Note the placement of the information command as the value of the punctuation key. This keeps it *within* the `\[ \]` delimiters.) Following Euler, this series can be transformed to the form

```
    \eval[p,S?=1]{\[
      \sum_{k=0}^\infty \frac{(-1)^k}{2^{k+1}}
      \sum_{n=0}^k(-1)^n\binom kn \frac1{(k-n)!}
    \]}[3] \nmcInfo{sum}.
```

$\Longrightarrow$

$$\sum_{k=0}^{\infty}\frac{(-1)^k}{2^{k+1}}\sum_{n=0}^{k}(-1)^n\binom{k}{n}\frac{1}{(k-n)!} = 0.368,$$

16 terms.
Note the setting `S?=1`. Without it, the summation stops after 1 term, the $k = 0$ term, because the $k = 1$ term vanishes. With `S?=1` it takes 16 terms of the *outer* sum to reach the stopping criterion. Since that sum starts at 0, that means that changing the upper limit from ∞ to 15 should give the same result – which it does – but it takes $\frac{1}{2} \times 16 \times 17 = 136$ terms in total to get there, to be compared with the 9 terms of the earlier simpler sum, and the terms are more complicated. Obviously such double sums are computationally intensive.

3.3 Changing default values

The settings option enables various settings to be changed for an individual calculation. You may find yourself wanting to make such changes sufficiently often that a change of default value is a better plan than encumbering each calculation with a list of settings.

Table 3.5: Default values, `\eval` command

key	value
rounding	`6`
pad	`0`
output-sci-notation	`0`
output-exponent-char	`e`
%	
input-sci-notation	`0`
input-exponent-char	`e`
multitoken-variables	`1`
logarithm-base	`10`
vv-display	`{,}\mskip 36mu minus 24mu(vv)`
vv-inline	`{,}\mskip 12mu plus 6mu minus 9mu(vv)`
%	
intify-rounding	`14`
%	
sum-extra-rounding	`2`
sum-query-terms	`0`
prod-extra-rounding	`2`
prod-query-terms	`0`

The way to do that is to create a *configuration file* with the name `numerica.cfg` in a text editor. Its entries, one per line, are of the form *key=value* followed by a comma, and for clarity preferably one entry per line (although this is not essential).The key names are noticeably more verbose than the corresponding keys of the settings option. The possible keys are listed in Table 3.5, together with their current default values.

Keys taking one of two possible values, `0` (for `false/off`) or `1` (for `true/on`), are `pad` (the result with zeros), `output-sci-notation`, `input-sci-notation`, and (check for) `multitoken-variables`.

The table is divided into four parts.

- The top four rows concern elements that can be changed for individual calculations with the trailing optional argument of `\eval`: rounding, padding with zeros, and outputting in scientific notation; see §2.3.
  - Note that to output the result always in scientific notation requires two settings, first setting `output-sci-notation` to `1`, and then choosing a character to act as the exponent marker. Because `l3fp` uses `e` for this character, `numerica` has made `e` its default. But this option is turned off by default (hence the `0` against this key).
- The next block of rows concern general elements that can be changed for individual calculations with the settings option of `\eval`; see §3.1. Obviously the key names are more expansive in the present context but the effect is the same.
  - But note that to input numbers in scientific notation requires two settings, first setting `input-sci-notation` to `1`, and then choosing a character to act as the exponent marker. Because `l3fp` uses `e` for this character, `numerica` has made `e` its default. The option is turned off by default (hence the `0` against this key).
- The third block is a single row specifying at what rounding value a floating point should be considered an integer; see §3.3.2 below.
- The last four rows concern default settings for infinite sums and products. These correspond to the keys `S+`, `S?` and `P+`, `P?` of the settings option that can be used to tweak the behaviour of the stopping criterion for such sums or products; see §3.2.

If you are dissatisfied with any of the default values listed, then in a text editor create a new file called `numerica.cfg` and assign *your* values to the relevant keys. For instance, if you find yourself working to 4 figures, that rounding to 6 is too many, then make the entry `rounding=`4. If also you want results always presented in proper scientific notation, $d.d_1d_2d_3d_4 \times 10^n$, then add a comma after `4` and enter on a new line (recommended but not strictly necessary; the comma is the crucial thing), `output-sci-notation=1,` (note the comma) and on another new line, `output-exponent-char=x`.

Perhaps you also want a non-zero setting for the final query terms for infinite sums and products. This makes sense if you are largely dealing with non-monotonic series – like Fourier series. Even the Euler transformation of the exponential series for e^{-1} discussed above required a non-zero `S?`. If you wish to make this change then add a comma and on a new line add (for instance) `sum-query-terms = 1,` and again on a new line, `prod-query-terms = 1`. If this is all you wish to change, then no comma is necessary after this final entry. Your newly created file should look something like

```
rounding             = 4,
output-sci-notation  = 1,
output-exponent-char = x,
sum-query-terms      = 1,
prod-query-terms     = 1
```

The white spacing may be different; white space is ignored by `numerica` when reading the file. Using it to align the equals signs helps *us* read the file. Note that the last entry, because it is the last entry, lacks a comma. Now save the file with the name `numerica.cfg`. This file will be read by `numerica` near the end of its loading process. These settings will be `numerica`'s defaults for the relevant keys.

3.3.1 Location of `numerica.cfg`

Save, yes, but where to? If the new settings are likely to apply only to your current document, then the document's directory is a sensible place to put it and `numerica` will certainly find it there since it is part of LaTeX3 file handling that file searches are not limited to the TeX distribution (including your personal texmf tree) but also include the current document directory. But what happens when you start working on another document? Will you remember to copy `numerica.cfg` to its new location? That is why your *personal texmf tree* is a better place.

3.3.1.1 Personal texmf tree?

This is a directory for 'waifs and strays' of the TeX system that are not included in the standard distributions like MiKTeX or TeXLive. Here you place personal packages designed for your own particular circumstances. These may include your own TeX or LaTeX package, say `mypackage.sty`, achieving some small or singular effect that doesn't warrant wider distribution on CTAN. Here you might place configuration files for other packages with your preferences (unless the package requires some specific location). Here you can put your personal bibliography files.

Your personal texmf tree is structured like the standard MiKTeX or TeXLive hierarchy but placed in another location so that there is no chance of its being overwritten when packages in MiKTeX or TeXLive are updated. But these distributions need to be alerted to its existence.

For example, in the MiKTeX console, click on Settings, and then on the Directories tab of the resulting dialog. Here you get to add your personal texmf hierarchy to the list of paths that MiKTeX searches, by clicking on the + button, browsing to your texmf folder and selecting it. By using the up and down arrow keys that the MiKTeX console provides, ensure that it lies *above* the the entry for the main MiKTEX tree. That way, files in your personal texmf tree will be found first and loaded. Now go to the Tasks menu and click on Refresh the filename database. This will let MiKTEX know what is held in your personal texmf tree. Files there can then be used like standard LATEX packages.

3.3.2 Rounding in 'int-ifying' calculations

Factorials, binomial coefficients, summation and product variables, and (in `numerica`) n-th roots from the `\sqrt` command, all require integer arguments. These integers may indeed be entered explicitly as integers, but they can also be determined as the result of a calculation. Rounding errors may mean the result is not an exact integer. How much leeway should be allowed before it is clear that the calculation did not give an integer result? In the default setup, `numerica` is generous. A number is considered an integer if it rounds to an integer when the rounding value is 14. Since `l3fp` works to 16 significant figures this provides more than enough 'elbowroom' for innocuous rounding errors to be accommodated. If a calculation does not round to an integer at a rounding value of 14 then it seems reasonable to conclude that it has *really* not given an integer answer, not just that rounding errors have accumulated. If you want to change this 'int-ifying' value for a particular calculation, then add a line to `numerica.cfg` like

```
intify-rounding = <integer>
```

Since `l3fp` works to 16 significant figures, values of `integer` greater than 16 are pointless. Generally int-ifying rounding values will be less than but close to 16 (although when testing the code I used some ridiculous values like 3 or 4). If other entries follow this one in the file, then conclude the line with a comma.

3.4 Parsing mathematical arguments

A main aim of the `numerica` package is to require minimal, preferably no, adjustment to the LATEX form in which an expression is typeset in order to evaluate it. But mathematicians do not follow codified rules of the kind programming languages insist on when writing formulas – like parenthesizing the arguments of functions, or inserting explicit multiplication signs (*) between juxtaposed terms. Hence the question of where the arguments of mathematical functions end is acute. For a few functions LATEX delimits the argument: think of `\sqrt`, `\frac`, `\binom`; also `^`. But for functions like `\sin` or `\tanh` or `\ln`, unary functions, this is not so. Nor is it for sums and products, and comparisons.

Before discussing the parsing rules for different groups of functions, I discuss the means `numerica` provides to handle exceptions to those rules, when one *does* need to make some adjustment to a formula.

3.4.1 The cleave commands `\q` and `\Q`

The word *cleave* has two opposed meanings: to adhere or cling to, and to split or sever. `numerica` defines two commands, `\q` and `\Q` to achieve these opposite effects. When a mathematical argument is being parsed, the `\q` command joins the next token to the argument (*cleaves to*); the `\Q` command severs the next token from the argument (*cleaves apart*). Neither command is added to the argument nor leaves a visible trace in the output.

Thus, without `\q`,

`\eval{$ \sin(n+\tfrac12)(x-t) $}[n=3,x=t+\pi,t=1.234]` $\Longrightarrow$
$\sin(n+\tfrac12)(x-t) = -1.102018,\ \ (n=3, x=t+\pi, t=1.234),$

which is $(\sin\frac72)\times\pi$. With `\q` between the bracketed factors,

`\eval{$ \sin(n+\tfrac12)\q(x-t) $}[n=3,x=t+\pi,t=1.234]` $\Longrightarrow$
$\sin(n+\tfrac12)(x-t) = -1,\ \ (n=3, x=t+\pi, t=1.234),$

which is $\sin(\frac72\pi)$. Similarly, without `\q`,

`\eval[p]{\[ \cos\frac{2\pi}{T}n(t+\tfrac12T) \]}[T=2,t=1,n=3]` $\Longrightarrow$

$$\cos\frac{2\pi}{T}n(t+\tfrac12T) = -6, \qquad (T=2, t=1, n=3),$$

which is $(\cos\pi)\times 3\times(1+\frac12\times 2)$. With `\q` used twice, once after the fraction and once before the left parenthesis,

`\eval[p]{\[ \cos\frac{2\pi}{T}\q n\q(t+\tfrac12T) \]}[T=2,t=1,n=3]`
$\Longrightarrow$

$$\cos\frac{2\pi}{T}n(t+\tfrac12T) = 1, \qquad (T=2, t=1, n=3),$$

which is $\cos(\pi\times 3\times 2)$.

It should be noted that for *trigonometric* functions, because of their use in Fourier series especially, there is another way of handling arguments with parentheses (and fractions). This is discussed in §3.4.2.3 below.

For the `\Q` command which splits an argument we have, without it,

`\eval{$ 1/2e $}` $\Longrightarrow 1/2e = 0.18394,$

which is the reciprocal of $2e$, whereas with the `\Q` command inserted before `e`,

`\eval{$ 1/2\Q e $}` $\Longrightarrow 1/2e = 1.359141,$

which is a half of e. Of course, the meaning in this example would be clearer if 1/2 were parenthesized or presented as a `\tfrac`.

3.4.1.1 Mnemonic

As mnemonic, best seen in sans serif for the Latin Modern fonts used in this document, think of the letter `q` as a circle *clinging* to a vertical descender; think of the letter `Q` as a circle *cut* by the diagonal stroke.

3.4.2 Parsing groups

The arguments of different groups of functions are handled in different ways. The criterion used for deciding when an argument ends for one group will not be that used for the others. Table §3.3.2 lists the different groups that `numerica` takes account of. At the top are functions or operations that have the smallest reach when determining where their arguments end; at the bottom are operations that have the greatest reach. The denominator of a slash fraction is treated as a unary function and is assigned to group II. By default trigonometric functions are treated the same as other unary functions but there is a setting which enables the direct (rather than inverse) trigonometric functions to accept a wider range of arguments, as occurs in Fourier series. Hence they are separated into their own group.

A formula is a sequence of tokens and brace groups. All parsing occurs from the left, LaTeX argument by LaTeX argument, where *argument* means either a token (an N-type argument in `expl3`-speak) or a brace group (an n-type argument). To distinguish LaTeX arguments from mathematical arguments I shall when necessary refer to L-args and M-args. A mathematical argument may end *at* an L-arg, meaning immediately before the L-arg, or end *with* the L-arg, meaning immediately after the L-arg. Ending or not will in general depend on whether the argument is in *first position* – the position immediately following a function token like `\sin` or `\log` – or in *general position* – any later position (although for trigonometric functions we will also need to consider *second* and even *third* position).

Table 3.6: Parsing groups

group	function/operation
I	surd, logical Not
II	unary functions, /
III	direct trig. functions
IV	sums, products
V	comparisons
VI	logical And, logical Or

For counting position, we need to allow for formatting elements and multi-token numbers – in both decimal and scientific formats. Formatting elements do not change the position count. This applies to things like thin spaces or phantoms (and their arguments) or modifiers like `\left` or `\biggl`. Multi-token numbers (in decimal or scientific formats) are treated as single items; they advance the position count by exactly one. LaTeX functions – like `\frac` – which take LaTeX arguments again advance the position count only by one. Mathematically, the fraction is viewed as a single unit.

I shall refer to a token or a token and its LaTeX arguments – like `\frac` and its arguments – as an *item*. Similarly, a (possibly multi-token) number is an item. Also it will help to distinguish tokens within brackets where both brackets lie to the right of a function from those that do not. The former I call *clothed*; the latter are *naked*. Thus the plus sign in $(\sin x + y)$ is naked relative to the sine (one bracket to the left of the function), but is clothed in $\sin(x + y)$ (both brackets to the right of the function).

3.4.2.1 Parsing group I

The only functions in this category are the surd and logical Not.

Why distinguish the surd from other unary functions? Surely we all agree that `\sin2\pi`, displaying as $\sin 2\pi$, vanishes? The argument of the sine extends beyond the 2 to include the π. But `\surd2\pi`, displaying as $\surd 2\pi$, is understood to be the product $\surd 2 \times \pi$. The argument of the surd ends with the 2. The surd binds more tightly to its argument than is true of unary functions generally.

For parsing group I

1. if a left bracket is in first position, the mathematical argument ends with the matching right bracket; otherwise
2. the argument ends with the item in first position and any L- or M-args required by that item.

If the factorial sign `!` *preceded* its argument, it too would belong to this parsing state, for it also binds tightly like the surd. This means that an expression like $\surd 4!$ is intrinsically ambiguous. Is it the square root of 24 or the factorial of 2? In `numerica` it produces the (perhaps rather odd) error

`\eval{$ \surd 4! $}` $\Longrightarrow$!!! Empty argument to fp-ify in: factorial. !!!

The surd has seized the argument; there is nothing for the factorial to operate on. The same error arises if the 4 is parenthesized, but parenthesizing like either `(\surd 4)!` or `\surd(4!)` repairs the situation. Because other unary functions (like the sine or logarithm) do not bind as tightly, this ambiguity does not arise for them.

Exponents cause no problem because taking square roots and raising to a power are commutative operations – the result is the same whichever is performed first.

`\eval{$ \surd 3^4 $}` $\Longrightarrow \surd 3^4 = 9.$

3.4.2.2 Parsing group II: unary functions, slash fractions

In the default setup this category includes the trigonometric and hyperbolic functions, their inverses, the various logarithms and the exponential functions, the signum function `\sgn`, and the slash fraction `/` where the argument to be determined is its denominator. Note however that there is a setting switch which enables trigonometric functions to handle parentheses in arguments more generally; see §3.4.2.3.

- In parsing group II we wish to accommodate usages like $\ln z^n = n\ln z$ (*HMF* 4.1.11), or $\operatorname{gd} z = 2\arctan e^z - \tfrac{1}{2}\pi$ (*HMF* 4.3.117), defining the Gudermannian. The exponent is included in the argument. Considering $\ln(1+1/n)^n$ exponents must also be part of parenthesized arguments.
- An approximation to Stirling's formula for the factorial is often written $\ln N! \approx N\ln N - N$ (widely used in texts on statistical mechanics). Hence the factorial sign should also be considered part of the argument.
- $\ln xy = \ln x + \ln y$ means the argument must reach over a product of variables. Identities like $\sin 2z = 2\sin z\cos z$ mean the argument also reaches over numbers, and expressions like $\sin\tfrac{1}{2}\pi x$ (*HMF* 4.3.104) mean that it further reaches over `\tfrac`-s and constants.
- Essentially *anything* can be in first position, and without parentheses; e.g.
  - unary functions: $\ln\ln z$ (*HMF* 4.1.52), $\ln\tan\dfrac{z}{2}$ (*HMF* 4.3.116),
  - fractions: $\ln\dfrac{z_1}{z_2}$ (*HMF* 4.1.9), $\arcsin\dfrac{(2ax+b)}{(b^2-4ac)^{1/2}}$ (*HMF* 3.3.36), $\ln\dfrac{\tan z}{z}$ (*HMF* 4.3.73),
  - absolute values: $\ln\left|\dfrac{a+x}{a-x}\right|$ (*HMF* 3.3.25),
  - square roots: $\arctan\sqrt{\dfrac{\nu_1}{\nu_2}F}$ (*HMF* 26.6.8)

With these examples in mind, for parsing group II

1. if a left bracket is in first position, the mathematical argument ends with the matching right bracket and any attached exponent, or factorial or double factorial sign; otherwise
2. the mathematical argument includes the item in first position and any L- or M-args required by that item;
   (a) if the item in first position is a number, variable, constant or `\tfrac`
      i. the argument appends the next item if it is a number, variable, constant or `\tfrac`, and so on recursively; or
      ii. the argument appends the next item if it is an exponent, or facorial or double factorial sign, and ends there; otherwise
      iii. the argument ends.
   (b) if the item in first position is not a number, variable, constant or `\tfrac`
      i. the argument appends the next item if it is an exponent, or factorial or double factorial sign, and ends there; otherwise
      ii. the argument ends.

An argument may extend over (see 2(a)i) numbers, constants, variables and `\tfrac`-s: $\sin 2\tfrac{p}{q}\pi x$ exhibits all elements.

Illustrating 1, the exponent is included in the argument but not the following variable:

`\eval{$ \log_{10}(1+2+3+4)^3n $}[n=5]` $\Longrightarrow$
$$\log_{10}(1+2+3+4)^3n = 15,\ \ (n=5).$$

For the sake of the reader, and as one naturally does in any case to avoid ambiguity, the formula should be written with the variable n preceding the logarithm: $n\log_{10}(1+2+3+4)^3$. The way the example is written suggests that the writer wished the n to be considered part of the argument. If that is the case, an outer set of parentheses would make intentions clear, but it is possible to leave the argument as written but insert a `\q` command before n:

`\eval{$ \log_{10}(1+2+3+4)^3\q n $}[n=5]` $\Longrightarrow$
$$\log_{10}(1+2+3+4)^3n = 3.69897,\ \ (n=5),$$

which is $\log_{10} 5000$.

Illustrating 2(a)ii, again the exponent is included in the argument but not the following variable:

`\eval{$ \log_{10}m^3n $}[m=10,n=5]` $\Longrightarrow$
$$\log_{10} m^3n = 15,\ \ (m=10, n=5).$$

Again, for the sake of the reader and as one naturally does to avoid ambiguity, the variable n should precede the logarithm. If in fact one wants the n included in the argument of the logarithm, the `\q` command is again available or, better in this case, the n can be shifted to precede the m, which illustrates 2(a)i:

`\eval{$ \log_{10}nm^3 $}[m=10,n=5]` $\Longrightarrow$
$$\log_{10} nm^3 = 3.69897,\ \ (m=10, n=5).$$

Is `numerica` being too strict when nm^3 is included in the argument of the logarithm, but m^3n is not? Any criterion is going to miss some instances where a different outcome might be desirable. Where an argument ends is affected by visual appearance. It is simple and easy to remember if it is understood that anything that breaks the visual appearance of juxtaposed numbers, variables, constants and `\tfrac`-s ends the argument. An exponent does just that.

Illustrating 2(b)ii, the argument stops with the `\dfrac` and its arguments and does not extend to the following constant:

`\eval{$ \sin\dfrac12\pi $}` $\Longrightarrow \sin\dfrac{1}{2}\pi = 1.50616.$

Obviously, someone writing an expression like this intends the π to be part of the argument. In that case, a `\tfrac` should be used. The `\dfrac` breaks the 'visual flow' of an argument.

Fractions

But why not a plain `\frac`? After all, for an inline expression it displays in the same way as a `\tfrac`. I considered making the argument-behaviour of `\frac` the same as `\tfrac` for text-style contexts, and the same as `\dfrac` for display-style contexts, but that would have meant the same expression evaluating to different results depending on the context, text-style or display-style, which ruled it out. Because `\frac` sometimes displays as `\dfrac`, it necessarily is treated like `\dfrac` (but see §3.4.2.3, specifically `()=2`).

Slash fractions

It is easy to write ambiguous expressions using the slash / to indicate fractions or division. How should $\pi/2n$ be interpreted? With from-the-left evaluation and calculator precedence rules which give equal precedence to * (multiplication) and / (division), this would be interpreted as $(\pi/2)\times n$, but most people will instinctively interpret it as $\pi/(2n)$. By placing `/` in parsing group II, this is what `numerica` does.

It treats the right-hand argument of the slash *as if it were the argument of a named function*. This means that $1/2\sin(\pi/6)$ is parsed as $(1/2)\sin(\pi/6)$ rather than as $1/(2\sin(\pi/6))$. It also means that $1/2\exp(1)$ and $1/2e$ give different results, which is acceptable since (in the author's view) they display differently and are not instinctively read in the same way.

3.4.2.3 Parsing group III

By default trigonometric functions are set to parsing group II. This accommodates many instances of how arguments are used with these functions, but Fourier series in particular require more. For them we need to take account of how *parentheses* are used in arguments. I find $\tan\tfrac{1}{2}(A+B)$ (*HMF* 4.3.148), $\sec\pi(\tfrac{1}{4}+\tfrac{1}{2}az)$ (*HMF* 19.3.3), $\cos(2m+p)z$ (*HMF* 20.2.3), $\sin(2n+1)v$ (*HMF* 16.38.1). Looking through various texts discussing Fourier series it is easy to find examples like

$$\cos\frac{2\pi}{T}nt,\quad \cos\frac{2\pi}{T}n(t+\tfrac{1}{2}T),$$

and

$$\cos(N+\tfrac{1}{2})\frac{2\pi\tau}{T},\quad \sin 2\pi\left(\frac{x}{\lambda}-\frac{t}{T}\right).$$

In the last of these `\left` and `\right` have been used to enlarge the parentheses.

All these usages can be accommodated by adjusting a setting in the settings option (§3.1) of the `\eval` command:

```
() = integer
```

where `integer` is one of `0, 1, 2`. For convenience of statement in what follows call parentheses, square brackets or braces *brackets*. If preceded by a `\left` or

`\right` or `\biggl` or `\biggr` etc. modifier, call them *Brackets*, with an uppercase 'B'. Modifiers do not contribute to the position count, so that a left Bracket in first position means the modifier and left bracket are both considered to be in first position. When it is immaterial whether it is a bracket or a Bracket I write b/Bracket. The rules that follow do not prescribe what mathematicians *ought* to do but are intended to be descriptive of certain patterns of mathematical practice as discerned in *HMF* and a number of texts (about half a dozen) on Fourier series.

`()=0` is the *default* setting; b/Brackets are included in the argument only if

- the left b/Bracket is in first position;
  - if the first item beyond the matching right b/Bracket is an exponent, or factorial or double factorial sign, it is appended to the argument, which ends there, otherwise
  - the argument ends with the right b/Bracket.

`()=1` includes a b/Bracketed expression in the argument, provided

- the left Bracket is in first position;
  - if the first item beyond the matching right Bracket is an exponent, or factorial or double factorial sign, it is appended to the argument, which ends there, otherwise
  - the argument ends with the right Bracket.
- or the item in first position is a number, variable, constant or `\tfrac` and the left bracket is in second position;
  - if the first item beyond the matching right bracket is an exponent, or factorial or double factorial sign, it is appended to the argument, which ends there, or
  - if the first item beyond the matching right bracket is a number, variable, constant, or `\tfrac` it is appended to the argument, and so on recursively, until
    - an exponent, or factorial or double factorial sign is met, which is appended to the argument which ends there, or
    - an item is met which is *not* an exponent, or factorial or double factorial sign, or a number, variable, constant or `\tfrac`, at which point the argument ends, or
    - the end of the formula is reached.

`()=2` includes a b/Bracketed expression in the argument provided

- the left b/Bracket is in first position, or the item in first position is a number, variable, constant, `\dfrac`, `\frac` or `\tfrac` and the left b/Bracket is in second position, or the items in first and second positions are numbers, variables, constants, `\dfrac`-s, `\frac`-s or `\tfrac`-s and the left b/Bracket is in third position;

- if the first item beyond the matching right b/Bracket is an exponent, or factorial or double factorial sign, it is appended to the argument, which ends there, or
- if the first item beyond the matching right b/Bracket is a number, variable, constant, `\dfrac`, `\frac` or `\tfrac` it is appended to the argument, and so on recursively, until
  * an exponent, or factorial or double factorial sign is met, which is appended to the argument which ends there, or
  * an item is met which is *not* an exponent, or factorial or double factorial sign, or a number, variable, constant, `\dfrac`, `\frac` or `\tfrac`, at which point the argument ends, or
  * the end of the formula is reached.

The default setting is `()=0` which is parsing group II behaviour.

`()=1` allows $\tan\frac12(A+B)$ and $\sec\pi(\frac14+\frac12 az)$, and $\cos(2m+p)z$ and $\sin(2n+1)v$, and also items on *both* sides of the bracketed part like $\sin\frac12(m+n)\pi$ provided there is only one item between the function and the left bracket:

`\eval[()=1]{$ \sin\tfrac16(m+n)\pi $}[m=1,n=2].` $\Longrightarrow$

$$\sin\tfrac16(m+n)\pi=1,\quad (m=1,n=2).$$

Note that `numerica` does not check what is included between the brackets – it could be anything. However inserting `\left`, `\right` modifiers before the parentheses restricts the argument of the sine in this example to the `\tfrac`:

`\eval[()=1]{$ \sin\tfrac16\left(m+n\right)\pi $}[m=1,n=2].` $\Longrightarrow$

$$\sin\tfrac16\left(m+n\right)\pi=1.563534,\quad (m=1,n=2).$$

`()=2` draws no distinction between brackets and Brackets. It allows all `()=1` possibilities but also *two* items (of a suitable kind) before the left b/Bracket; it also allows `\dfrac`-s and `\frac`-s in addition to `\tfrac`-s.

The following examples are taken from different texts on Fourier series. The first shows a `\frac` being included in the argument, the second shows *two* items – including a `\frac` – preceding the left parenthesis, the third shows a `\frac` to the right of the parentheses, and the fourth shows parentheses using `\left-\right` modifiers with two items preceding them:

$$\cos\frac{2\pi}{T}nt,\quad \cos\frac{2\pi}{T}n(t+\tfrac12T),\quad \sin(N+\tfrac12)\frac{2\pi\tau}{T}\quad\text{and}\quad \sin2\pi\left(\frac{x}{\lambda}-\frac{t}{T}\right).$$

All these usages are accommodated by the `()=2` setting. For instance

```
\eval[p,()=2]
  {
    \[ \sin(N+\tfrac12)\frac{2\pi\tau}T \]
  }[N=1,\tau=2,T=3]
```

$\Longrightarrow$

$$\sin(N+\tfrac{1}{2})\frac{2\pi\tau}{T}=0,\qquad (N=1,\tau=2,T=3),$$

which is the sine of $(\frac{3}{2})\times(\frac{4}{3}\pi)=2\pi$ (and *not* $(\sin\frac{3}{2})(\frac{4}{3}\pi)$), where a `\frac` trailing the parentheses has been included in the argument. Or consider

```
\eval[p,()=2]
  {\[
     \sin2\pi\left(\frac{x}{\lambda}
        -\frac{t}{T}\right)
  \]}[x=1,\lambda=2,t=3,T=4]
```

$\Longrightarrow$

$$\sin 2\pi\left(\frac{x}{\lambda}-\frac{t}{T}\right)=-1,\qquad (x=1,\lambda=2,t=3,T=4),$$

which is the sine of $2\pi\times(-\frac{1}{4})=-\frac{1}{2}\pi$ (and *not* $\sin 2\pi$ times the parenthesised expression) where there are two items before the parentheses which surround two `\frac`-s and `\left` and `\right` modifiers have been used with the parentheses.

However a usage like $\sin(n+\frac{1}{2})(x-t)$, noted in two different texts, is not available without explicit use of the `\q` command between the parenthesized groups.

3.4.2.4 Parsing group IV

The only members of this group are `\sum` and `\prod`.

For parsing group IV

1. the argument ends
   (a) at the first naked plus or minus sign encountered, or
   (b) at the first comparison sign or comparison command encountered, or
   (c) at the first logical And or logical Or sign encountered, or
   (d) at the end of the formula.

In practice this means mainly (a) and (d), and seems to be the instinctive practice. *HMF* has multiple examples in multiple chapters of the argument to a sum ending at a naked plus sign: 7.3.12 & 7.3.14, 9.1.11 & 9.1.77, 9.6.35 & 9.6.43, 11.1.9, ... (at that point I stopped looking). They were all of the form

$$\sum \text{argument} + \ldots$$

A minus sign serving the same purpose was harder to find but *HMF* 10.4.65 & 10.4.67 are two instances. I considered whether a `\times` or slash fraction sign `/` might end the argument of a sum, but surely we need to allow things like $\sum 1/n^2$ which rules out the slash and *HMF* 9.9.11 provides two of a number of instances in *HMF* of sum arguments continuing past explicit `\times` signs (at line breaks when a summand spills onto a second line).

Because they are evaluated using the same code as sums I (unthinkingly) placed products with sums but doubts later intruded. In *HMF* products occur only occasionally and are almost all of the form

$$\prod(\text{argument})$$

where the argument is bracketed (often with `\left \right` modifiers) and the multiplicand ends with the right bracket. At least twice (*HMF* 6.1.25 and 24.2.2.1) an exponent (-1) is attached to the right bracket and the argument ends there. Looking further afield, a text on number theory has examples where the argument of the product extends to *three* parenthesised factors, $\prod(\text{arg1})(\text{arg2})(\text{arg3})$ and a number of others where it extends to two. A text on theory of functions has

$$\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)e^{z/n}$$

although *HMF*, for the same expression, encloses the two factors within (large) square brackets, as if some ambiguity existed as to how far the reach of the `\prod` extended.

Tentatively I retain products here in the same group as sums.

3.4.2.5 Parsing group V

Comparison symbols compose this group: `=`, `<`, `>`, `\ne`, `\le`, `\ge`, `\leq`, `\geq`, and the various comparison commands from the `amssymb` package listed in §2.3.4.6. Because of the way `numerica` handles comparisons, it is the argument on the right-hand side of the relation that needs determining.

For parsing group V

1. the argument ends at
   (a) the first logical And or logical Or encountered, or
   (b) the first comparison sign or command encountered, or
   (c) the end of the formula.

3.4.2.6 Parsing group VI

Logical And and logical Or are the sole members of this group. It is the right-hand side of the And or Or command that needs determining.

For parsing group VI

1. the argument ends at
   (a) the first logical And or logical Or encountered, or
   (b) the end of the formula.

3.4.2.7 Disclaimer

The parsing rules of the different groups are not normative; they are not statements of how mathematical formulas should be written. Rather they are attempts to discern regularities in how mathematicians often do write formulas. It is how things look in the pdf, not LaTeX, that is the guide. You are always free to parenthesize as you see fit and to insert cleave commands (`\q` or `\Q`) to force outcomes.

(But note that parenthesizing has its limits. For sums, writing

$$\sum (< \texttt{stuff} >) < \texttt{more} - \texttt{stuff} >$$

does not necessarily end the summand at the right parenthesis: it ends at the first naked $+$ or $-$ sign, or `\Q` command, encountered.)

The rule should always be to write expressions that are clear to the reader of the pdf. An expression that is ambiguous to the reader, even if it fits within the parsing rules, is to be deplored. The *intent* is that `\eval` can parse unambiguous expressions correctly.

Chapter 4

Nesting `\eval` commands

`\eval` commands can be used within `\eval` commands, both as part of the formula being evaluated or as part of the vv-list or both. Only in special circumstances is this likely to be useful – perhaps when a calculation can be divided into two or more parts where different settings are appropriate for the different parts. One can imagine cases in which trigonometric functions are involved and different `()` settings would be helpful in different parts of the formula. Nesting of command within command becomes especially significant with the additional commands available when `numerica` is loaded with the `plus` or `tables` options; see the associated documents `numerica-plus.pdf` and `numerica-tables.pdf`. Since those additional commands are not available in this document, I restrict myself here to some 'toy' examples of `\eval` commands within `\eval` commands to show how things work.

4.1 Star option for inner `\eval`

The `\eval` command 'digests' a LaTeX formula to produce an `l3fp`-readable formula. This is then fed to `l3fp` to be evaluated. The evaluated output is then formatted in various ways to be displayed. If the inner `\eval` command produces formatted output, it is *that* that the outer `\eval` command will attempt to digest – and fail. Hence *always* use the star option for the inner `\eval` command. That means the outer `\eval` is feeding on a number only:

`\eval{$ \sin(\eval*{\sin x}[x=\pi/6]\pi) + 1 $}` $\Longrightarrow \sin(0.5\pi) + 1 = 2.$

Also no math delimiters are used in the inner command. These are irrelevant with the star option in any case, but in the present context would cause error if included because they would be treated as part of the formula and thereby produce an 'unknown token' error message. In the presentation of the overall result that the inner `\eval` command is evaluated, showing as 0.5.

4.2 Nesting in the vv-list

`\eval{$ \sin k\pi + 1 $}[k=\eval*{\sin x},x=\pi/6]` $\Longrightarrow$
$\sin k\pi + 1 = 2, \;\; (k = 0.5, x = \pi/6).$

When the inner `\eval` command is in the vv-list of the outer command and has a vv-list of its own, then the entire inner command needs to be placed in braces:

`\eval{$ \sin k\pi + z $}[k={\eval*{y\sin x}[x=\pi/4,y=1/\surd2]},z=1]`
$\Longrightarrow \sin k\pi + z = 2, \;\; (k = 0.5, z = 1).$

The vv-list of the inner `\eval` command contains both a comma and square brackets. Both elements need to be hidden from the outer `\eval` in order that *its* vv-list be parsed correctly. Hence braces surround the inner `\eval` and its arguments. The same need arises if the inner `\eval` has a non-empty settings option – another comma-separated square-bracketed option.

The values of variables used in an inner `\eval` command are restricted to that command; they do not 'leak' into the outer calculation. But variables and their values in the outer vv-list are available for use in the inner `\eval` command (unless a value is explicitly changed in the inner vv-list).

4.2.1 Debugging

It is worth looking at the debug display when `\eval` commands are nested. For the outer `\eval` command:

`\eval[dbg=210]{$ \sin \eval*{\sin x}[x=\pi/6]\pi + 1 $}` $\Longrightarrow$

```
 vv-list:
formula:  \sin \eval *{\sin x}[x=\pi /6]\pi + 1
  stored:
fp-form:  sin((0.5)(pi))+1
  result:  2
```

and when the inner `\eval` is in the vv-list,

`\eval[dbg=210]{$ \sin k\pi + 1 $}[k=\eval*{\sin x},x=\pi/6]` $\Longrightarrow$

```
 vv-list:  x=\pi /6, k=\eval *{\sin x}
formula:  \sin k\pi + 1
  stored:  x=0.5235987755982988, k=0.5
fp-form:  sin((0.5)(pi))+1
  result:  2
```

For the inner `\eval` command debugging still works but in an idiosyncratic way. To clarify exactly what is going on I have added a `\left( \right)` pair around the entire inner `\eval` command. Note that I have also used a *negative* `dbg` value. With a positive value, the right parenthesis is pressed toward the right margin of the page. The negative value limits the display to the text width and gives the much neater result shown.

```
\eval[()=2]{$
  \sin\left(
        \eval*[dbg=-210]{ \sin x }[x=\pi/6]
      \right)\pi + 1 $}
```

$$\Longrightarrow \sin\left(\begin{array}{rl} \text{vv-list:} & \text{x=\textbackslash pi /6} \\ \text{formula:} & \ \text{\textbackslash sin x} \\ \text{stored:} & \text{x=0.5235987755982988} \\ \text{fp-form:} & \text{sin((0.5235987755982988))} \\ \text{result:} & \text{0.5} \end{array}\right)\pi+7=8$$

The debug display from the inner `\eval` command has been inserted into the formula of the outer `\eval` in the position occupied by the inner `\eval`. I did not deliberately code for this, but have decided to leave it as is, since there can be no confusion about which `\eval` command is being 'debugged', despite the potential for some rather odd displays. In this last example, in order to both use `\left(...\right)` and have the calculation give the previous result I have employed the setting `()=2` in the outer `\eval`; see §3.4.2.3.

Chapter 5

\nmcInfo (\info)

Used after the evaluation of an 'infinite' process, the `\nmcInfo` command, or its equivalent short-name form `\info` will tell you how many terms or factors or iterations or steps were needed to arrive at the result. The syntax of the `\nmcInfo` command is

```
\nmcInfo{<arg>}
```

where `<arg>` is restricted to one of two choices at present, either `sum` or `prod`. If the package `numerica-plus.def` is loaded (see §1.1.1 and the associated document `numerica-plus.pdf`) two further arguments are possible: `iter` and `solve`.

There is a starred form of the command:

```
\nmcInfo*{<arg>}
```

(or `\info*{<arg>}`). As with the `\eval` command the star has the effect of suppressing anything other than the numerical result from the display.

As an example, let's test a standard identity, $\cosh^2 x - \sinh^2 x = 1$, 'the hard way'. We know that $\cosh x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$ and $\sinh x = x \prod_{k=1}^{\infty} \left(1 + \frac{x^2}{k^2\pi^2}\right)$. The difference of their squares should be 1:

```
\eval{\[
  \left[\sum_{n=0}^{\infty}
    \frac{x^{2n}}{(2n)!}
  \right]^2-
    \left[x\prod_{k=1}^{\infty}
      \left(1+\frac{x^{2}}{k^{2}\pi^{2}}\right)
    \right]^2
  \]}[x=1][3] \info{sum}\quad \info{prod}
```

$\Longrightarrow$

$$\left[\sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}\right]^2 - \left[x \prod_{k=1}^{\infty} \left(1 + \frac{x^2}{k^2\pi^2}\right)\right]^2 = 1.002, \qquad (x = 1)$$

5 terms   119 factors.

Nearly right. Obviously the product converges only slowly which is where the error comes from (see the discussion in §3.2, where we needed the extra rounding setting `P+=3` and 350 factors to get a correct 3-figure value). The point of the example is to show the information command being used for both sum and product in the one evaluation. One does not exclude the other.

The information command can also be placed in the settings option as the value of the punctuation setting. An example of this has already been provided earlier which I'll repeat here:

```
\eval[p=\mbox{,\quad\nmcInfo{sum}.}]
  {\[ \sum_{n=0}^{\infty}\frac{(-1)^{n}}{n!} \]}[3]
```

$\Longrightarrow$

$$\sum_{n=0}^{\infty}\frac{(-1)^{n}}{n!}=0.368,\quad \text{9 terms.}$$

Because of the `\[ \]` delimiters, if the information command had been placed *after* the `\eval` command, it would have slid down to the next line. As it is, it resides *inside* the `\[ \]` delimiters, on the same line as the expression. This may be significant for adjusting vertical spacing of subsequent elements of the document.

5.1 Errors

Should the wrong argument be used in the `\nmcInfo` command, no harm is done:

```
\eval{$
  \sum_{k=0}^{\infty}\binom \alpha k x^k
    $}[x=1/2,\alpha=3], \ \info{prod}
```

$\Longrightarrow \sum_{k=0}^{\infty}\binom{\alpha}{k}x^k=3.375,\ \ (x=1/2,\alpha=3)$, 119 factors.

119 *factors*? The information command is remembering a previous result, the last time `prod` was used as its argument. Changing the argument from `prod` to `sum` reveals the correct number of *terms*.

Should a non-existent argument be used, an error message is generated:

```
\eval{$
  \sum_{k=0}^{\infty}\binom \alpha k x^k
    $}[x=1/2,\alpha=3], \\ \info{Fred}
```

$\Longrightarrow \sum_{k=0}^{\infty}\binom{\alpha}{k}x^k=3.375,\ \ (x=1/2,\alpha=3)$,
!!! Unknown argument `Fred` in: info command. !!!

Chapter 6

Saving and reusing results

You may want to use at some place in a document a result calculated earlier. It would be good to be able to do so without having to do the calculation again at the new location. `numerica` offers a command `\nmcReuse` which saves a result to a control sequence which can be used elsewhere in the document, expanding to the saved result. The control sequence and its content are also saved to file for use on other occasions.

The syntax of `\nmcReuse` is simple. The command takes two optional arguments, a star (asterisk) and a conventional square-bracket delimited argument. If both are used it looks like

```
\nmcReuse*[csname]
```

where `csname` is the proposed name of what will become the control sequence `\csname` containing the latest result from the `\eval` command. The name should be composed of letters only.

As with `\nmcEvaluate` and `\nmcInfo` there is a short-name form, `\reuse`, for `\nmcReuse`. If a conflict with another package arises, it should be possible to fall back on `\nmcReuse`.

6.1 Use without optional argument: `\nmcReuse`

Suppose your document is `mydoc.tex` (so that the LaTeX command `\jobname` expands to `mydoc`). If `\nmcReuse` is used without optional arguments, then `numerica` checks for the existence of a file `mydoc.nmc` in the current document directory and if found loads and records the contents of `mydoc.nmc`. The contents should be a comma separated list of control sequences and braced values like `\csname1 {value1},\csname2 {value2},...` The control sequences can then be used elsewhere in the document. In particular, control sequences containing numerical values can be used in expressions within `\eval` commands and in vv-lists.[1]

Note that the control sequences are given LaTeX definitions using `xparse`'s `\NewDocumentCommand` (a little more general than LaTeX 2_ε's `\newcommand`) and can be used at any later point in the document simply by entering the control sequence (e.g., `\csname1`) there (but see §6.2.1). Should there already be a control sequence with the same name, LaTeX will generate an error and halt compilation.

6.2 Use with optional name: `\nmcReuse[csname]`

If a name *is* supplied, say `\nmcReuse[csname]`, then not only does `numerica` first look for `mydoc.nmc` (assuming your document is called `mydoc.tex`) and load the values stored in that file if they have not already been loaded, but it also defines `\csname` to contain the latest result from the `\eval` command. Should `\csname` already be present in `mydoc.nmc` and so have been loaded with the other values from `mydoc.nmc`, the old value is overwritten with the new value using `xparse`'s `\RenewDocumentCommand` and the new value is saved to the file `mydoc.nmc`.

6.2.1 Group level

Control sequences like `\csname` defined by `\nmcReuse` are defined within the current group level. A usage like

```
$ \eval{1+1}\reuse[two] $
```

confines the definition of `\two` to the environment delimited by the dollar signs. A usage like

```
$ \eval{1+1} $ \reuse[two],
```

where the command has been moved outside the math delimiters, still confines the definition of `\two` to whatever larger environment the `\reuse` command might lie within. If in fact the definition occurs at document level then `\two` (in the present example) is available for use throughout the document, otherwise it is available only within the confines of the current environment.

This is likely to be *not* what is wanted. The remedy is simple: precede the saved control sequence – `\two` in the present instance – with a 'naked' `\reuse` command. This loads the value stored in `mydoc.nmc`. In the following example, `\reuse` is used within a math environment which is followed by some text (`blah blah result:`) then a 'naked' `\reuse` command and the control sequence `\two` (between math delimiters, which are necessary for reasons explained in the next section).

`$ \eval{1+1} \reuse[two] $, blah blah result: \reuse $\two$.` $\Longrightarrow$
2, blah blah result: 2.

[1]The associated document `numerica-plus.pdf` describes how other quantities like tables and sequences of numerical values (iterates, recurrences) can also be saved and reused.

6.3 What is saved?

In the default configuration `\nmcReuse` saves the entirety of the display resulting from the latest `\eval`-uation. This may include invisible formatting elements meaning that what one expects to be only a number cannot be inserted into text without causing a LaTeX error; it requires a math environment to print – see the last example.

The simplest way to avoid this awkwardness is to use `\eval` with the star option. This produces a numerical result with *no* formatting. In the following example, I calculate `11+11` with `\eval*` and store the value in the control sequence `\twos`. I then enter some text (`Blah blah blah:`) and insert the control sequence `\twos` into the text without math delimiters. As you can see, `\twos` has expanded to 22, the result of the calculation.

`\eval*{$ 11+11 $}. \nmcReuse[twos] Blah blah blah: \twos` $\Longrightarrow$ 22.
Blah blah blah: 22

6.3.1 Viewing what is saved: `\reuse*`

To view what is saved in the `.nmc` file append a star (asterisk) to the `\reuse` command. (This makes particular sense when using `numerica` in a program like LyX with a preview facility; see Chapter 7.)

`\reuse*` $\Longrightarrow$

Saved: \iandi {2},\two {2\mathchoice {}{}{}{}},\twos {22}

Particularly notable here is the invisible formatting `\mathchoice{}{}{}{}` accompanying `2` in the value of `\two`.

The two options of the`\reuse` command can be used together, in which case the control sequence resulting from the name supplied in the square-bracketed option will appear in the list resulting from the star option, i.e. the list contains not just what has been saved earlier but also the current control sequence saved.

6.3.2 `\eval`'s `reuse` setting

The star option of the `\eval` command allows a purely numerical result to be saved, but also only a number is displayed. By using the `reuse` setting of the `\eval` command it is possible to have both a full display of an evaluation, vv-list and all, and to save only a numerical result.

For the *starred* form of the `\eval` command it is always *only the numerical result* that is saved, whatever the value of the `reuse` key in the settings option of the `\eval` command.

For the *unstarred* form of the `\eval` command exactly what is saved with `\nmcReuse` depends on the `reuse` setting:

```
reuse = <integer>
```

where `<integer>` can take one of two values,

- `reuse=0` (the default) saves *the form that is displayed* including a formatting component. If the result is displayed in the form *formula=result (vv-list)* then that is what is saved; if the display is of the form *result (vv-list)* then that is what is saved; if the vv-list is empty, an empty formatting component is still present in the saved result;
- `reuse=1` (or, indeed, any non-zero integer) saves only the numerical result with no other elements of the display (meaning no formatting component).

Thus, with the default setting (`reuse=0`) the full content of the display is saved:

`\eval{$ x + y $}[x=1,y=1] \reuse[iandi]` $\Longrightarrow x+y=2,\ \ (x=1,y=1)$.

To check that this is the case, `\reuse \iandi` $\Longrightarrow x+y=2,\ \ (x=1,y=1)$.

On the other hand, with `reuse=1` only the numerical value is saved:

`\eval[reuse=1]{$ x + y $}[x=1,y=1] \reuse[iandi]` $\Longrightarrow$
$x+y=2,\ \ (x=1,y=1)$,

which we can check here: `\reuse\iandi` $\Longrightarrow$ 2.

6.3.2.1 `\reuse` in the preamble

To gain access from the outset to the control sequences stored in the file `mydoc.nmc`, place `\nmcReuse` without an optional argument in the preamble (but after `\usepackage{numerica}`).

Chapter 7

Using `numerica` with LYX

The document processor LYX has a facility that enables snippets from a larger document to be compiled separately and the results presented to the user without having to compile the entire document. The present document was written in LYX. The demonstration calculations were evaluated using this *instant preview* facility.

To use `numerica` in LYX go to Document ▷ Settings ▷ LaTeX Preamble and enter

```
\usepackage{numerica}
```

then click OK. However preview poses problems for the straightforward use of the `\nmcReuse` command. If you wish to use this command in LYX then `numerica` should be loaded with the `lyx` package option. Thus in Document ▷ Settings ▷ LaTeX Preamble enter

```
\usepackage[lyx]{numerica}
```

then click OK, or you may wish to follow the above line in the preamble with `\nmcReuse`,

```
\usepackage[lyx]{numerica}
\nmcReuse
```

and *then* click OK. The additional line ensures all saved values are available in your document from the outset.

7.1 Instant preview

Preview performs localised mini-LATEX runs on selected parts of a document (for instance, the mathematical parts) and displays the results in LYX while the user continues to work on the surrounding document. `numerica` uses these local LaTeX runs to do its evaluations and display their results. That means you get feedback on your calculations almost immediately.

To use this facility first ensure that instant preview is turned on. This means selecting Tools ▷ Preferences ▷ Look & Feel ▷ Display and against Instant preview selecting On, then clicking OK.

7.1.1 Conflict with hyperref support?

There may be a conflict in LyX between hyperref support and preview, not that the previews do not form but that their formation takes a circuitous path that noticeably slows their display.[1] If this occurs on your system, go to Document ▷ Settings ▷ PDF Properties and ensure the check box Use Hyperref Support is cleared. By all means reset the check box when you come finally to compile your document, but until then it should result in a noticeably brisker display of previews if the checkbox is cleared.

7.2 Mathed

(Mathed = the LyX mathematics editor.) If you have instant preview *on* then one way to use `numerica` in LyX is to enter an `\eval` command in mathed. Clicking the cursor outside the editor with the mouse or moving it outside with the arrow keys will then trigger formation of a preview of the editor's contents – a snippet of what will be shown in the pdf. This will be displayed in mathed's place after a generally short 'pause for thought' as the mini-LaTeX run progresses behind the scenes.

The original expression can be recovered by clicking on the preview. The content of mathed is immediately displayed and can be edited.

7.2.1 LaTeX braces { }

LyX does not support `numerica`'s `\eval` command 'out of the box' as it does, say, `\frac` or `\sqrt`. To use the `\eval` command in mathed you will need to supply the braces used to delimit its mandatory argument. (For `\frac` and `\sqrt` by contrast, LyX supplies these automatically.) Unfortunately the `{` key[2] does not insert a left brace into the document but rather an escaped left brace `\{` as you can see by looking at View ▷ Code Preview Pane. Escaped braces like this are used for grouping terms in *mathematics*; they are not the delimiters of a LaTeX argument.

The brace delimiters for LaTeX arguments are entered in mathed by typing a backslash \ then { (two separate key presses rather than a single combined press). This enters a balanced pair of (unescaped) braces with the cursor sitting between them waiting for input. Alternatively, if you have already written an expression that you want to place between braces, select it, then type \ then {.

[1] At least there is on the author's Windows 10 system, but I'm not sure that this is general.
[2] Shift+[on my keyboard.

7.3 Preview insets vs mathed

There are problems with using mathed for calculations.

- Expressions entered in mathed are necessarily of the form `$ \eval... $` or more generally `delimiter \eval... delimiter`. But you may wish to wrap the `\eval` command *around* the math delimiters to produce a *formula=result* form of display. In mathed the only way to do that is to write the *formula=* part yourself – which may involve no more than copy and paste but is still additional mouse work/key pressing.
- Mathed does not accept carriage returns. If you want to format a complicated expression for readability by breaking it into separate lines, you can't. The expression is jammed into the one line, along with the settings option content and the vv-list.

For these reasons I have come to prefer *not* using mathed for calculations but instead to use preview insets wrapped around TEX-code (ERT) insets. LYX uses the shortcut Ctrl+L to insert an ERT inset. Since LYX now does no printing itself, the shortcut Ctrl+P that was formerly used for printing is available for other purposes. On my keyboard, the P key lies diagonally up and to the right but adjacent to the L key. I suggest assigning Ctrl+P to inserting a preview inset. Then typing Ctrl+P Ctrl+L – which means holding the Ctrl key down and tapping two adjacent keys, P followed immediately by L – will insert an ERT inset inside a preview inset with the cursor sitting inside the ERT inset waiting for input. In the ERT inset you can enter carriage returns, and so format complicated expressions. You can place the vv-list on a separate line or onto consecutive lines. And when you have finished, clicking outside the preview inset will trigger preview into doing its thing and present the result 'before your eyes'.

To assign the suggested shortcut, go to Tools ▷ Preferences ▷ Editing ▷ Shortcuts. Under Cursor, Mouse and Editing Functions in the main window on the right, scroll down until you come to preview-insert, select it, then click Modify. Now press Ctrl+P. The shortcut will magically appear in the greyed, depressed key. Click OK and then OK in the Preferences window to close it. (Most of the examples in this document have been evaluated in this way, using Ctrl+P Ctrl+L.)

7.4 Errors

Instant preview will display `numerica` error messages in LYX just as it does the results of calculations. Clicking on the message will show the underlying expression which can then be edited. However LATEX errors will *not* produce a preview; formation of the preview will stall. To find precisely what has gone wrong, you will need to look at the LATEX log, but not the log of the overall document; rather the *preview* log. Unfortunately this is tucked away in a temporary directory and is not immediately accessible in LYX (unlike the main LATEX log from Document ▷ LaTeX Log). When LyX is started, it sets up a temporary directory in which to perform various tasks. On Windows systems this will be located in `C:\Users\<your name>\AppData\Local\Temp` and will have a name like `lyx_tmpdir.XOsSGhBc1344`.

One of the tasks LyX uses this temporary directory for is to create preview images when a document is opened. If you look inside LyX's temporary directory when a document is first loaded, you will see a subdirectory created, with a name like `lyx_tmpbuf0`. There may already be such directories there, in which case the number on the end will be greater than `0` – it depends on whether other documents are or have been open in the current instance of LyX. Inside the appropriate `lyx_tmpbuf`*n* folder will be the preview log with a name like `lyxpreviewZL1344.log`. It will usually be accompanied by other files with extensions like `.dvi`, `.tex`, and perhaps quite a number with the extension `.png`, each one of which is a preview, or part of a preview. For a document just loaded there will be only the one preview log, but if you have added preview insets or math insets to your document in the current editing session there will be a number of such logs and you will need to determine the relevant one by the time stamp.

The log files are text files and can be opened in a text editor. The relevant part of the log is towards the end (just before the final statistical summary) where you will find a list of entries like `Preview: Snippet 1 641947 163840 7864588`. If there is an error, it will be noted here among these snippets and will generally make clear what needs remedying.

7.4.1 CPU usage, LaTeX processes

It is possible when a preview fails to resolve that the LaTeX process associated with the preview will continue to run, using CPU cycles, slowing overall computer performance, and perhaps resulting in extra fan use giving a different sound to the computer. In Windows 10, the Task Manager (Ctrl+Shift+esc) under the Details tab shows the current executables running. The CPU column will show which processes are preoccupying the CPU. Check whether one or more of these processes looks LaTeX-related (e.g. `latex.exe` or `pdflatex.exe`, or `miktex-pdftex.exe` if using MiKTeX). Click the Name column to sort the processes by name and look for the relevant name in the list, select it, and end the process (click the End Task button).

I am not familiar with the corresponding situation on Linux or Mac.

7.5 Using `\nmcReuse`

As noted, LyX creates its previews in a temporary directory, not the document directory. If you want to save values from your current document – say, `mydoc.lyx` – to `mydoc.nmc` then you can do so without drama, but `mydoc.nmc` will be located in the temporary directory, and when LyX is closed will be deleted along with the temporary directory.

Suppose first that at the end of a session you manually copy `mydoc.nmc` back to the document directory. How can you ensure that the values saved in this file are available the next time you open `mydoc.lyx`? As noted at the start of this chapter entering

```
\usepackage[lyx]{numerica}
\nmcReuse
```

in the preamble ensures that these saved values are available for use from the outset – available to the mini-LaTeX runs creating previews in the temporary directory.

That leaves the problem of saving new values from the current session, which are stored in `mydoc.nmc` in the *temporary* directory, back to `mydoc.nmc` in the *document* directory. When LyX is closed the temporary directory with all its contents is deleted. As suggested already we could manually copy `mydoc.nmc` from the temporary directory to the document directory but that means remembering to do so before closing LyX. Inevitably we will sometimes forget.

Fortunately LyX has a copying mechanism for getting files out of the temporary directory. When a document is exported – say to pdf – it is possible to specify a *copier* to automatically copy back to the document directory or subdirectory various files in the temporary directory. We want the `.nmc` file containing the saved values to be copied back. Go to Tools ▷ Preferences ▷ File Handling ▷ File Formats and find PDF (pdflatex) (assuming export to `pdf` by this route) in the list of formats. In the Copier slot of the dialogue insert the following line of code:

```
python -tt $$s/scripts/ext_copy.py -e nmc,pdf -d $$i $$o
```

`ext_copy.py` is a python script that is supplied with LyX. The `-e nmc,pdf -d` part of the line tells `ext_copy.py` that on export to `pdf` by the `pdflatex` route to copy any files with the extensions `.nmc` or `.pdf` from the temporary directory where LyX does its work back to the document directory – the `-d` option (which became available from LyX 2.3.0).

But if you have a complex document, it may take too much time to want to export to pdf before closing LyX, particularly if there are a lot of evaluations in the document. Much faster is to export to *plain text*, not because you want a plain text version of your document but because it too can be used to trigger the copier mechanism. Go to Tools ▷ Preferences ▷ File Handling ▷ File Formats and find Plain text in the list of formats. In the Copier slot enter

```
python -tt $$s/scripts/ext_copy.py -e nmc -d $$i $$o
```

The only difference from the previous copier command is the absence of `pdf`.[3] This will copy `mydoc.nmc` with its saved values from the temporary directory

[3] I'm assuming that you don't actually want the plain text version of the file copied back. If you do, then change `-e nmc` to `-e nmc,txt`.

back to the document directory. To effect the export, go to File ▷ Export and find Plain text in the list of formats and click on it.

A shortcut would be nice. For that go to Tools ▷ Preferences ▷ Editing ▷ Shortcuts, click on New, enter `buffer-export text` in the Function: slot, click on the blank key against Shortcut: and type your shortcut. You may have to try a number before you find one that hasn't already been assigned. (I'm using Ctrl+; for no particular reason beyond the fact that it fits under the fingers easily and saving values to the document directory has a punctuation-like feel to it, a pause in the process of writing.) It is now an easy matter to press the shortcut at the end of a LYX session to copy all the values saved in `mydoc.nmc` back to a file of the same name in the document directory. And it is brisk, not least because plain text export ignores ERT insets (and hence preview insets wrapped around ERT insets), nor does it evaluate `\eval` commands in math insets.

7.5.1 A final tweak?

But one still needs to *remember* to press the shortcut. The thought arises: can *closing* the current document trigger the copying process? LYX provides a means of linking two commands and assigning a keyboard shortcut to them with its `command-sequence` LYX function. I suggest assigning a shortcut to

```
command-sequence buffer-export text; view-close
```

Indeed, why not reassign the current shortcut for `view-close`, which is Ctrl+W on my system, to this command sequence? (I use the `cua` key bindings – check the Bind file: slot in Tools ▷ Preferences ▷ Editing ▷ Shortcuts.)

Please note, however, that *this will work as intended only from LYX 2.4.0.*[4] For LYX 2.3 and earlier, the command sequence will generally fail because of 'asynchronous' processing – `buffer-export` and `view-close` use different threads and the latter may well start before the former is complete. From LYX 2.4.0 this defect has been fixed. You press your shortcut, the export to plain text occurs and the `.nmc` file is copied back to the document directory, then the current view is closed.

7.6 Using LYX notes

The central fact about a LYX note is that it does not contribute to the pdf. But instant preview still works there. This suggests a possibility: that a calculation be performed within a LYX note and the result saved using `\nmcReuse` within the same preview inset. The saved value is now available *from file* for use elsewhere in the document. In this way, some selected content from a LyX note *can* find its way into the pdf when the document is compiled.

[4]Due for release in the first half of 2021.

Chapter 8

Reference summary

8.1 Commands defined in `numerica`

1. `\nmcEvaluate, \eval`
2. `\nmcInfo, \info,`
3. `\nmcReuse, \reuse`
4. `\q, \Q` (‘cleave’ commands)

Provided they have not already been defined when `numerica` is loaded, the following commands are defined in `numerica` using `\DeclareMathOperator` from `amsmath` :

1. `\arccsc, \arcsec, \arccot`
2. `\csch, \sech`
3. `\asinh, \acosh, \atanh, \acsch, \asech, \acoth`
4. `\sgn, \lb`

Provided they have not already been defined, the following commands are defined in `numerica` using `\DeclarePairedDelimiter` from `mathtools`:

```
\abs, \ceil, \floor
```

The following commands have been redefined in `numerica` to give more spacing around the underlying `\wedge` and `\vee` symbols:

```
\land, \lor
```

8.2 'Digestible' content

`numerica` knows how to deal with the following content, meaning that any of these elements occurring within an `\eval` command should not of itself cause a `numerica` error. Not all formatting commands affect display of the output.

1. variable names (sequences of tokens given values in the variable = value list)
2. digits, decimal point
   (a) `1, 2, 3, 4, 5, 6, 7, 8, 9, 0, .`
3. constants
   (a) `e, \pi, \gamma, \phi, \deg, \infty` (sometimes)
4. arithmetic operators
   (a) `+, -, *, /, ^, \times, \cdot, \div`
5. logical operators
   (a) `\wedge, \land, \vee, \lor, \neg, \lnot`
6. comparisons
   (a) `=, <, >, \ne, \neq, \le, \leq, \ge, \geq`
   (b) (if `amssymb` loaded) `\nless, \ngtr, \geqq, \geqslant, \leqq, \leqslant, \ngeq, \ngeqq, \ngeqslant, \nleq, \nleqq, \nleqslant`
7. brackets, bracket-like elements, modifiers
   (a) `( ), [ ], \{ \}`
   (b) `\lparen \rparen` (from `mathtools`), `\lbrack \rbrack, \lbrace \rbrace`
   (c) `\lvert \rvert, \lfloor \rfloor, \lceil \rceil`
   (d) `| |` (no nesting, deprecated)
   (e) `\left \right, \bigl \bigr, \Bigl \Bigr, \biggl \biggr, \Biggl \Biggr`
   (f) `. / |` (used with a modifier)
   (g) `\abs[]{}, \abs*{}, \floor[]{}, \floor*{}, \ceil[]{}, \ceil*{}`
8. unary functions (in the mathematical sense)
   (a) `\sin, \cos, \tan, \csc, \sec, \cot`
   (b) `\arcsin, \arccos, \arctan, arccsc, \arcsec, \arccot`
   (c) `\sin^{-1}, \cos^{-1}, \tan^{-1}, \csc^{-1}, \sec^{-1}, \cot^{-1}`

(d) `\sinh, \cosh, \tanh, \csch, \sech, \coth`

(e) `\asinh, \acosh, \atanh, \csch, \sech, \acoth`

(f) `\sinh^{-1}, \cosh^{-1}, \tanh^{-1}, \csch^{-1}, \sech^{-1}, \acoth^{-1}`

(g) `\exp, \lb, \lg, \ln, \log, \log_{}, \sgn, \surd`

(h) `\sqrt{}, \abs[]{}, \abs*{}, \floor[]{}, \floor*{}, \ceil[]{}, \ceil*{}`

(i) `!, !!` (prepended argument)

9. binary functions

   (a) `\tfrac{}{}, \frac{}{}, \dfrac{}{}`

   (b) `\tbinom{}{}, \binom{}{}, \dbinom{}{}`

   (c) `\sqrt[]{}`

10. *n*-ary functions

    (a) `\min, \max, \gcd`

11. sum, prod

    (a) `\sum_{}^, \prod_{}^`

12. formatting commands

    (a) , (comma, in *n*-ary functions)

    (b) `{}, \\, &, \to`

    (c) `\dots, \ldots, \cdots,`

    (d) `\ , \,, \;, \:, \!, \>`

    (e) `\thinspace, \quad, \qquad , \hfill, \hfil`

    (f) `\phantom{}, \vphantom{}, \hphantom{}`

    (g) `\xmathstrut[]{} , \splitfrac{}{}, \splitdfrac{}{}` (from `mathtools`), `\mathstrut`

    (h) `\displaystyle, \textstyle, \scriptstyle, \scriptscriptstyle`

    (i) `\label{}, \ensuremath{}, \text{}, \mbox{}`

    (j) `\begin{}, \end{}`

13. font commands

    (a) `\mathrm{}, \mathit{}, \mathcal{}, \mathtt{}, \mathbf{}, \mathbb{}, \mathsf{}, \mathfrak{}, \mathscr{}, \mathnormal{}, \boldsymbol{}`

8.3 Settings

8.3.1 Available `\nmcEvaluate` settings

key	type	meaning	default
`dbg`	int	debug 'magic' integer	`0`
`^`	char	exponent mark for sci. notation input	`e`
`xx`	int (0/1)	multi-token variable switch	`1`
`()`	int (0/1/2)	trig. function arg. parsing	`0`
`o`		degree switch for trig. funcions	
`log`	num	base of logarithms for `\log`	`10`
`vvmode`	int (0/1)	vv-list calculation mode	`0`
`vvd`	token(s)	vv-list display-style spec.	`{,}\mskip 12mu plus 6mu minus 9mu(vv)`
`vvi`	token(s)	vv-list text-style spec.	`{,}\mskip 36mu minus 24mu(vv)`
`*`		switch to suppress equation numbering (if `\\` in `vvd`)	
`p`	char(s)	punctuation (esp. in display-style)	`,`
`S+`	int	extra rounding for stopping criterion, sums	`2`
`S?`	int ≥ 0	query stopping with these final terms, sums	`0`
`P+`	int	extra rounding for stopping criterion, products	`2`
`P?`	int ≥ 0	query stopping with these final terms, products	`0`
`reuse`	int	form of result saved with `\nmcReuse`	`0`

8.3.2 Available configuration file settings

key	default
`rounding`	`6`
`pad`	`0`
`output-sci-notation`	`0`
`output-exponent-char`	`e`
`input-sci-notation`	`0`
`input-exponent-char`	`e`
`multitoken-variables`	`1`
`logarithm-base`	`10`
`intify-rounding`	`14`
`vv-display`	`{,}\mskip 36mu minus 24mu(vv)`
`vv-inline`	`{,}\mskip 12mu plus 6mu minus 9mu(vv)`
`sum-extra-rounding`	`2`
`sum-query-terms`	`0`
`prod-extra-rounding`	`2`
`prod-query-terms`	`0`
`eval-reuse`	`0`