`numerica-plus`

Andrew Parsloe
(ajparsloe@gmail.com)

August 22, 2023

Abstract

The `numerica-plus` package defines commands to iterate functions of a single variable, find fixed points of such functions, find the zeros or extrema of such functions, and calculate the terms of recurrence relations.

Note:

- This document applies to version 3.0.0 of `numerica-plus.`
- Version 3 of `numerica` is required; (`numerica` requires `amsmath` and `mathtools`).
- I refer many times in this document to *Handbook of Mathematical Functions*, edited by Milton Abramowitz and Irene A. Stegun, Dover, 1965. This is abbreviated to *HMF*, and often followed by a number like 1.2.3 to locate the actual expression or value referenced.
- Version 3.0.0 of `numerica-plus`
  - is compatible with the additional features of numerica version 3.0.0,
  - including the decimal comma if the `comma` package option is used with numerica;
  - amends and adds to documentation, including
  - a section on finding roots with Newton-Raphson iteration.

Contents

Chapter 1

Introduction

Entering

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

in the preamble of your document makes available the commands

- `\nmcIterate`, a command to iterate a function (apply it repeatedly to itself), including finding fixed points (values of x where $f(x) = x$);
- `\nmcSolve`, a command to find the zeros of functions of a single variable (values of x for which $f(x) = 0$) or, failing that, local maxima or minima of such functions;
- `\nmcRecur`, a command to calculate the values of terms in recurrence relations in a single recurrence variable (like the terms of the Fibonacci sequence or Legendre polynomials).

A main difference from version 2 of `numerica-plus` is that the package now does not load `numerica` automatically but, rather, requires the user to explicitly call `numerica` (with options if wanted). Version 3 of `numerica` is required, and must be loaded *before* `numerica-plus`. With `numerica` loaded you get access to the commands `\nmcEvaluate` (`\eval`), `\nmcInfo` (`\info`), `\nmcMacros` (`\macros`), `\nmcConstants` (`\constants`), and `\nmcReuse` (`\reuse`); see the `numerica` documentation for details on the use of these commands. The `numerica` package options, and particularly the `comma` and `rounding` options specifying the decimal comma and default rounding value, apply in `numerica-plus`.

The commands of the present package all share the syntax of `\nmcEvaluate`. I will discuss them individually in later chapters but turn first to a meaningful example to illustrate their use and give a sense of ‘what they are about’.

1.1 Example of use: the rotating disk

Consider a disk rotating uniformly with angular velocity ω in an anticlockwise sense in an inertial system in which the disk's centre **0** is at rest. Three distinct points **1**, **2**, **3** are fixed in the disk and, in a co-rotating polar coordinate system centred at **0**, have polar coordinates (r_i, θ_i) $(i, j = 1, 2, 3)$. Choose **01** as initial line so that $\theta_1 = 0$.

The cosine rule for solving triangles tells us that the time t_{ij} in the underlying inertial system for a signal to pass from point **i** to point **j** satisfies the equation

$$t_{ij} = c^{-1}\sqrt{r_i^2 + r_j^2 - 2r_i r_j \cos(\theta_j - \theta_i + \omega t_{ij})} \equiv f(t_{ij}),$$

where c is the speed of light and $i, j \in \{1, 2, 3\}$. (Equally, we could be describing an acoustic signal between points on a disk rotating uniformly in a still, uniform atmosphere – in which case c would be the speed of sound.) Although the equation doesn't solve algebraically for the time t_{ij}, it does tell us that $t = t_{ij}$ is a *fixed point* of the function f. To calculate fixed points we use the command `\nmcIterate`, or its short-name form `\iter`, with the star option, `\iter*`. For `\iter` the star option means: continue iterating until a fixed point has been reached and, as with the `\eval` command, suppress all elements from the display save for the numerical result.

First, though, values need to be assigned to the various parameters. Suppose we use units in which $c = 30$, and $\omega = 0.2$ radians per second. To avoid having to write these values in the vv-list every time, I have put in the preamble to this document the statement

```
\constants{ c=30,\omega=0.2 }
```

For the polar coordinates of **1** and **3** I have chosen $r_1 = 10$, $r_3 = 20$ and $\theta_3 = 0.2$ radians (remember $\theta_1 = 0$). To find a fixed point t_{13} I give t an initial trial value 1 (plucked from the air). Its position as the *rightmost* item in the vv-list tells `\iter` that t is the iteration variable:

```
\iter*{ c^{-1}\sqrt{r_1^2+r_3^2-2r_1 r_3
          \cos(\theta_3+\omega t)}
      }[ r_1=10,r_3=20,\theta_3=0.2,t=1 ],
\quad\info{iter}.
```

$\Longrightarrow$ 0.356899, 5 iterations. The short-name form of the `\nmcInfo` command from `numerica` has been used to display the number of iterations required to attain the fixed-point value.

To six figures, only five iterations are needed, which seems rapid but we can check this by substituting $t = 0.356899$ back into the formula and `\eval`-uating it:

```
\eval*{ c^{-1}\sqrt{r_1^2+r_3^2-2r_1 r_3
          \cos(\theta_3+\omega t)}
      }[ r_1=10,r_3=20,\theta_3=0.2,t=0.356899 ]
```

$\Longrightarrow$ 0.356899, confirming that we have indeed calculated a fixed point. That it did indeed take only 5 iterations can be checked by omitting the asterisk from the `\iter` command and specifying the total number of iterations to perform. I choose `do=7` to show not just the 5th iteration but also the next two just to confirm that the result is stable. We shall view all 7: `see=7`. Because of the length of the formula I have suppressed display of the vv-list by giving the key `vv` an empty value:

```
\iter[do=7,see=7,vv=]
  {\[ c^{-1}\sqrt{r_1^2+r_3^2-2r_1 r_3
    \cos(\theta_3+\omega t)} \]}
      [ r_1=10,r_3=20,\theta_3=0.2,t=1 ]
```

$\Longrightarrow$

$$\begin{aligned} c^{-1}\sqrt{r_1^2+r_3^2-2r_1r_3\cos(\theta_3+\omega t)} &= 0.382355 \\ &\hookrightarrow 0.357756 \\ &\hookrightarrow 0.356928 \\ &\hookrightarrow 0.3569 \\ &\hookrightarrow 0.356899 \\ &\hookrightarrow 0.356899 \\ &\hookrightarrow 0.356899 \end{aligned}$$

The display makes clear that on the 5th iteration, the 6-figure value has been attained.

Alternatively, we could use the `\nmcRecur` command (or its short-name form `\recur`) to view the successive iterations, since an iteration is a first-order recurrence: $f_{n+1} = f(f_n)$:

```
\recur[env=multline*,vv={,}\\(vv)\\,
              do=8,see1=0,see2=5]
  { f_{n+1}=c^{-1}\sqrt{r_1^2+r_3^2-2r_1 r_3
    \cos(\theta_3+\omega f_{n})} }
      [ r_1=10,r_3=20,\theta_3=0.2,f_{0}=1 ]
```

$\Longrightarrow$

$$\begin{multline*} f_{n+1}=c^{-1}\sqrt{r_1^2+r_3^2-2r_1r_3\cos(\theta_3+\omega f_n)}, \\ (r_1=10, r_3=20, \theta_3=0.2, f_0=1) \\ \to\ 0.356928,\ 0.3569,\ 0.356899,\ 0.356899,\ 0.356899 \end{multline*}$$

I have specified `do=8` terms rather than 7 since the zero-th term ($f_0 = 1$) is included in the count. I've chosen to view the last 5 of them but none prior to those by writing `see1=0,see2=5`. Note the choice of environment and the `vv` setting, pushing display of the vv-list and result to new lines and suppressing equation numbering with the `*` on the `multline`.

Another and perhaps more obvious way to find the value of t_{13}, is to look for a zero of the function $f(t) - t$. That means using the command `\nmcSolve` (or its short-name form `\solve`). I shall do so with the star option `\solve*` which suppresses display of all but the numerical result. A trial value for t is required. I have chosen `t=0`:

```
\solve*{ c^{-1}\sqrt{r_1^2+r_3^2-2r_1 r_3
  \cos(\theta_3+\omega t)} - t }
    [ r_1=10,r_3=20,\theta_3=0.2,t=0 ],
\quad \nmcInfo{solve}.
```

$\Longrightarrow$ 0.356898, 1+20 steps.

Nearly the same answer as before is attained but this time many more steps have been required. This is to be expected. The `\solve` command uses the bisection method, finding an interval where the function has opposite signs at the end points and successively bisecting it to locate the zero between. Since $1/2^{10} \approx 1/10^3$, about 10 bisections are needed to determine 3 decimal places. Hence we can expect about 20 bisections for a 6-decimal-place answer. The particular form of the `\nmcInfo` command display, '$1 + 20$ steps', indicates that it took 1 search step to find an interval where the function had opposite signs at the end points and, within that interval, 20 bisections to narrow the position of the zero to 6-figures. I will discuss the discrepancy in the final figure in Chapter 3; see §3.3.1.3.

1.1.1 Circuits

Okay, so we can calculate the time taken in the underlying inertial system for a signal to pass from one point of the rotating disk to another. How long does it take to traverse the circuit **1231**, i.e. a signal from **1** to **2** to **3** and back to **1**? That means forming the sum $t_{12} + t_{23} + t_{31}$, which means calculating the separate t_{ij} and then using `\eval` to calculate their sum.

To simplify things, I assume a little symmetry. Let the (polar) coordinates of **1** be $(a, 0)$, of **2** be $(r, -\theta)$, and of **3** be (r, θ): **2** and **3** are at the same radial distance from the centre **0** and at the same angular distance from the line **01** but on opposite sides of it, **3** ahead of the line, **2** behind it. The rotation is in the direction of positive θ. Rather than just calculate $t_{12} + t_{23} + t_{31}$ for the circuit **1231**, I also calculate the time $t_{13} + t_{32} + t_{21}$ for a signal to traverse the same circuit but in the opposite sense, **1321**, and compare them (form the difference).

Note that with **2** and **3** positioned as they are relative to **1**, a signal against the rotation from **3** to **1** takes the same time as a signal from **1** to **2** and, in the sense of rotation, a signal from **2** to **1** takes the same time as a signal from **1** to **3**, so that the round trip times are $2t_{12} + t_{23}$ and $2t_{13} + t_{32}$.

To see this, suppose the signal from **2** to **1** starts at time $t = 0$; it reaches **1** at a later time t' when the disk has rotated through an angle $\omega t'$. Viewed from the underlying inertial system, the signal path is a straight line from point **2** $(r, -\theta)$ to point **1** $(a, \omega t')$ subtending an angle $\theta + \omega t'$ at the centre **0**. But **3** $(r, \theta + \omega t')$ at time t' and **1** $(a, 0)$ at time $t = 0$ also subtend an angle $\theta + \omega t'$ at **0** in the underlying inertial system. In the underlying inertial system the line segments $\mathbf{1}_0\mathbf{3}_{t'}$ and $\mathbf{2}_0\mathbf{1}_{t'}$ are of equal length (indeed reflections of each other in the bisector of the arc $\mathbf{1}_0\mathbf{1}_{t'}$) so that $t_{13} = t_{21}$. Similarly, if a signal from **3** at time $t = 0$ reaches **1** at time $t = t''$ then $\mathbf{3}_0\mathbf{1}_{t''}$ and $\mathbf{1}_0\mathbf{2}_{t''}$ are of equal length and $t_{31} = t_{12}$.

1.1.1.1 Nesting commands

Analytically, both t_{21} and t_{13} are the same fixed point of the function of t

$$c^{-1}\sqrt{r^2 + a^2 - 2ra\cos(\theta + \omega t)}$$

and t_{31} and t_{12} are the same fixed point of the function of t

$$c^{-1}\sqrt{r^2 + a^2 - 2ra\cos(\theta - \omega t)}.$$

To calculate $2t_{12} + t_{23}$ therefore means calculating

```
2\iter*{ c^{-1}\sqrt{a^2+r^2-2ar
              \cos(\theta-\omega t)} }
  + \iter*{ c^{-1}\sqrt{2r^2-2r^2
              \cos(2\theta+\omega t)} }
```

with the analogous expression for $2t_{13} + t_{32}$. But we can do the comparison of round trip times 'in one go' by nesting the `\iter*` commands inside an `\eval*` command:

```
\eval*{ % circuit 1231
    2\iter[var=t]{ c^{-1}\sqrt{a^2+r^2-2ar
              \cos(\theta-\omega t)} }[8]
  + \iter[var=t]{ c^{-1}\sqrt{2r^2-2r^2
              \cos(2\theta+\omega t)} }[8]
% circuit 1321
  - 2\iter[var=t]{ c^{-1}\sqrt{a^2+r^2-2ar
              \cos(\theta+\omega t)} }[8]
  - \iter[var=t]{ c^{-1}\sqrt{2r^2-2r^2
              \cos(2\theta-\omega t)} }[8]
      }[ a=10,r=20,\theta=0.2,t=1 ]
```

$\Longrightarrow$ 0.034746 .

By itself this result is of little interest beyond seeing that `numerica-plus` can handle the calculation. What *is* interesting is to find values of our parameters for which the time difference vanishes – say values of θ, given the other parameters and especially the value of r. Is there a circuit such that it takes a signal the same time to travel in opposite senses around the circuit, despite the rotation of the disk? Rather than nesting the `\iter` commands inside an `\eval`, we need to nest them in a `\solve` command:

```
\solve[env=multline*,p=.,var=\theta,+=1]
  {% circuit 1231
    2\times\iter[var=t,+=1]{ c^{-1}\sqrt{a^2+r^2-2ar
              \cos(\theta-\omega t)} }
  + \iter[var=t,+=1]{ c^{-1}\sqrt{2r^2-2r^2
              \cos(2\theta+\omega t)} }
  % circuit 1321
  - 2\times\iter[var=t,+=1]{ c^{-1}\sqrt{a^2+r^2-2ar
              \cos(\theta+\omega t)} }
  - \iter[var=t,+=1]{ c^{-1}\sqrt{2r^2-2r^2
              \cos(2\theta-\omega t)} }
  }[ a=10,r=20,\theta=0.1,t=1 ][4]
```

$\Longrightarrow$

$$2\times 0.5378+1.2213-2\times 0.6144-1.068=0,\qquad (a=10,r=20,\theta=0.1,t=1)\ \rightarrow\ \theta=1.0358.$$

One point to note here is the use of `\times` (in `2\times\iter`). In this example the formula is displayed – because of the use of the `env` setting, `env=multline*`. Without the `\times` the result would have been the same but the display of the formula would have juxtaposed the '2's against the following decimals, making it look as if signal travel times were 20.5378 and 20.6144 (and no doubt causing perplexity). The unfamiliar settings are discussed in the relevant chapters below.

So this expression gives a value of $\theta_{\Delta t=0}$ for one value of r. The obvious next step is to create a table of such values, which can be done with the `\tabulate` command from the associated package `numerica-tables` wrapped around an expression like the one above. (For my own interest I have done this. On a High St laptop it is not fast – plenty of time to make a nice hot cup of tea.) But this is not a research paper on the rotating disk. I wished to show how the different commands of `numerica-plus` can be used to explore a meaningful problem. And although it looks as if a lot of typing is involved, once $c^{-1}\sqrt{r^2+a^2-2ra\cos(\theta-\omega t)}$ has been formed in LaTeX and values specified in the vv-list (and the `\constants` command in the preamble), much of the rest is copy-and-paste with minor editing.

1.2 Shared syntax of the new commands

`numerica-plus` offers three new commands for three processes: `\nmcIterate` (short-name form `\iter`) for iterating functions, `\nmcSolve` (short-name form `\solve`) for finding the zeros or (local) extrema of functions, and `\nmcRecur` (short-name form `\recur`) for calculating terms of recurrence relations. All three commands share the syntax of `\nmcEvaluate` (or `\eval`) detailed in the associated document `numerica.pdf`. When all options are used the command looks like, for instance,

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

You can substitute `\nmcSolve`, or `\nmcRecur` for `\nmcIterate` here. The arguments are the same as those for `\nmcEvaluate`.

1. `*` optional switch; if present ensures a single number output with no formatting, or an appropriate error message if the single number cannot be produced;
2. `[settings]` optional comma-separated list of *key=value* settings for this particular command and calculation;
3. `{expr.}` the only mandatory argument; the mathematical expression in LaTeX form that is the object of interest;
4. `[vv-list]` optional comma-separated list of *variable=value* items; for `\iter` and `\solve` the *rightmost* (or innermost) variable in the vv-list may have special significance;
5. `[num. format]` optional format specification for presentation of the numerical result (rounding, padding with zeros, scientific notation); boolean output is suppressed for these commands.

The way the result is displayed follows the same pattern as for `\nmcEvaluate` (see the associated document `numerica.pdf`), depending on whether a math environment wraps around a command, is wrapped within a command, is invoked with the `env` setting, or is completely absent. And with version 3 of `numerica-plus`, as in `numerica`, there is always the `f` setting which turns on (`f=1`) and turns off (`f=0`) display of the formula. These matters are irrelevant for the starred forms of commands, which give number-only results. Looking at the various examples in the preceding section on the rotating disk you will see illustrations of many of these different situations.

As well as the `f` setting, most of the settings available to the `\eval` command are also available to the present commands although not all will be relevant or have effect.[1] Refer to the associated document `numerica.pdf` for a discussion of such settings. The 'trick' in version 2 of `numerica` and `numerica-plus` of

---

[1] In particular the `ff` (multi-formula) setting has not, as yet, been implemented in version 3.0.0 of `numerica-plus` (in the interest of getting the whole revised `numerica` suite launched).

converting environments delimited by `\[` and `\]`) into `multline` environments whenever the vv-list specification included a newline character (`\\`) has been dispensed with. Now, if you want a `multline` or other environment, directly specify it with the `env` setting. The enhanced handling of environments in `numerica` version 3 means the `*` setting which was available in earlier versions of `numerica` and `numerica-plus` to suppress equation numbering is now unnecessary and has been removed (although its presence will not cause an error – only leave a message in the log file and on the terminal). Starring the environment in the standard LaTeX way, `env=equation*`, `env=multline*`, etc., does the job.

In addition to the inherited settings, each of the `numerica-plus` commands has settings of its own, discussed at the relevant parts of the following chapters.

Section 1.1 also provides examples of commands being nested. Nesting may occur not only in the main argument of a command, but also in the vv-list, or even the settings option. (The associated document `numerica.pdf` has an example of the last possibility.)

Chapter 2

Iterating functions: `\nmcIterate`

Only in desperation would one try to evaluate a continued fraction by stacking fraction upon fraction upon fraction like so:

```
\eval{\[ 1+\frac{1}{1+\frac{1}{1+\frac{1}
  {1+\frac{1}{1+\frac{1}{1+\frac{1}
    {1+\frac{1}{1+\frac{1}{1+\frac{1}
      {1+\frac{1}{1+\frac{1}
        {1+\frac{1}{1}}}}}}}}}}}} \]}
```

$\Longrightarrow$

$$1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1}}}}}}}}}}}} = 1.618026$$

`numerica-plus` provides a command for tackling problems like this sensibly. In such problems a function is repeatedly applied to itself (iterated). This is done through the command `\nmcIterate` or (short-name form) `\iter`.

2.1 Basic use

To evaluate the continued fraction we want to apply $1+1/x$ to itself repeatedly. So, we wrap `\iter` around `1+1/x` and give `x` an initial value `1` in the vv-list:

`\iter{\[ 1+1/x \]}[x=1]` $\Longrightarrow$

$$\begin{aligned}
&1+1/x=2, \qquad (x=1)\\
&\qquad\quad \hookrightarrow 1.5\\
&\qquad\quad \hookrightarrow 1.666667\\
&\qquad\quad \hookrightarrow 1.6\\
&\qquad\quad \hookrightarrow 1.625
\end{aligned}$$

This hints that it might be heading in the direction of $\frac{\sqrt{5}+1}{2} = 1.618034$ but clearly not enough iterations have been performed to confirm this. By default, the `\iter` command performs 5 evaluations, the initial evaluation producing in this instance the result 2, which then becomes the new value of x, producing the first iteration proper and giving the value 1.5, which in turn becomes the new value of x and so on. Also by default, `\iter` displays the first evaluation and the results of the final 4.

Both these numbers, 5 and 4, are likely to be too small. They can be easily changed with the `do` and `see` settings. Increasing the number of iterations in the example to `do=17` and the displayed results to `see=5` shows how the iteration of $1 + 1/x$ indeed stabilizes at 1.618034:

`\iter[do=17,see=5]{\[ 1+1/x \]}[x=1]` $\Longrightarrow$

$$\begin{aligned}
&1+1/x=2, \qquad (x=1)\\
&\qquad\quad \ldots \text{ final 5 of 17:}\\
&\qquad\quad \hookrightarrow 1.618037\\
&\qquad\quad \hookrightarrow 1.618033\\
&\qquad\quad \hookrightarrow 1.618034\\
&\qquad\quad \hookrightarrow 1.618034\\
&\qquad\quad \hookrightarrow 1.618034
\end{aligned}$$

But iteration of functions is not limited to continued fractions. Particularly since the emergence of chaos theory, iteration has become an important study in its own right. Any function with range within its domain can be iterated – repeatedly applied to itself. The cosine is an example:

`\iter[do=20]{\[ \cos x \]}[x=\pi/2]` $\Longrightarrow$

$$\begin{aligned}
&\cos x=0, \qquad (x=\pi/2)\\
&\qquad \ldots \text{ final 4 of 20:}\\
&\qquad \hookrightarrow 0.738369\\
&\qquad \hookrightarrow 0.739567\\
&\qquad \hookrightarrow 0.73876\\
&\qquad \hookrightarrow 0.739304
\end{aligned}$$

which displays the initial value and last four of 20 evaluations of $\cos x$ when the initial value of x is $\frac{\pi}{2}$. It looks as if the cosine is ‘cautiously’ approaching a limit, perhaps around 0.738 or 0.739. We need to nearly double the number of iterations to confirm that this is so (to the default 6 figures):

`\iter[do=39]{\[ \cos x \]}[x=\pi/2]` $\Longrightarrow$

$$\begin{array}{l}\cos x = 0, \qquad (x = \pi/2)\\ \quad\ldots\text{ final 4 of 39:}\\ \quad\hookrightarrow 0.739086\\ \quad\hookrightarrow 0.739085\\ \quad\hookrightarrow 0.739085\\ \quad\hookrightarrow 0.739085\end{array}$$

The display is largely fixed, hard-coded, at least at this stage. It uses an `array` environment. Whether it is centred or treated in an inline manner depends on whether displaystyle or inline (or no) delimiters are used. Rather than using explicit delimiters, the `env` setting can also be used (there are examples below).

As already noted, for a function to be iterated indefinitely, its range must lie within or be equal to its domain. If even part of the range of a function lies outside its domain then on repeated iteration there is a chance that a value will eventually be calculated which lies in this 'outside' region. Iteration cannot continue beyond this point and an error message is generated. As an example consider the inverse cosine, `\arccos`. This can be iterated only so far as the iterated values lie between ±1 inclusive. If we try to iterate `\arccos` at 0 for example, since $\cos\frac{1}{2}\pi = 0$, $\arccos 0 = 1.5708$ $(\frac{1}{2}\pi)$ so only a first iterate is possible. But we could choose an initial value more carefully; 37 iterations of the cosine at $\frac{1}{2}\pi$ led to a fixed point 0.739085, so let's choose 0.739085 as initial point and perform 37 iterations:

`\iter[do=37]{\[ \arccos x \]}[x=0.739085]` $\Longrightarrow$

$$\begin{array}{l}\arccos x = 0.739085, \qquad (x = 0.739085)\\ \quad\ldots\text{ final 4 of 37:}\\ \quad\hookrightarrow 0.644659\\ \quad\hookrightarrow 0.870219\\ \quad\hookrightarrow 0.515149\\ \quad\hookrightarrow 1.029615\end{array}$$

The result of the 37th iteration is greater than 1. Thus increasing the number of iterations to 38 should generate an error message:

`\iter[do=38,see=4]{\[ \arccos x \]}[x=0.739085]` $\Longrightarrow$!!! `l3fp` error 'Invalid operation' in: formula. !!!

which it does. `l3fp` objects when asked to find the inverse cosine of a number greater than 1.

2.1.1 Logistic map

The logistic map came to prominence with a 1976 paper by the biologist Robert May. He examined the equation

$$x_{n+1} = rx_n(1 - x_n),$$

where $x_n \in [0,1]$ and represents the ratio of an existing population to the maximum possible population. The intent is to capture two opposed effects: reproduction, with the rate of population increase proportional to the population when the population is small ($x_{n+1} \approx rx_n$), and mortality, when the population approaches the 'carrying capacity' of the environment ($x_{n+1} \approx r(1-x_n)$).

The logistic map $rx(1-x)$ exhibits a variety of behaviours depending on the value of $r \in (0,4)$. The Wikipedia article on the subject lists the following:

1. With $0 \leq r \leq 1$, the population dies, independent of the initial population.

2. With $1 \leq r \leq 2$, the population will quickly approach the value $1-1/r$, independent of the initial population.

3. With $2 \leq r \leq 3$, the population will also eventually approach the same value $1-1/r$, but first will fluctuate around that value for some time. The rate of convergence is linear, except for $r=3$, when it is dramatically slow.

4. With r between 3 and $1+\sqrt{6} \approx 3.44949$ the population will eventually oscillate between two values $x_\pm = \big(r+1 \pm \sqrt{(r-3)(r+1)}\,\big)\big/2r$.

5. With r between 3.44949 and 3.54409 (approximately), from almost all initial conditions the population eventually oscillates among four values.

6. With r beyond 3.54409, from almost all initial conditions, the population eventually oscillates among 8 values, then 16, then 32, and so on. The lengths of the intervals of r for each oscillation regime decrease rapidly; the ratio between the lengths of successive bifurcation intervals approaches the Feigenbaum constant $\delta \approx 4.66920$. This is an example of a period-doubling cascade.

7. $r \approx 3.56995$, at the end of the period-doubling cascade, marks the onset of chaos. Most values of r beyond this, from almost all initial conditions, no longer yield periodic oscillations. Slight variations in the initial population value yield significantly different results over time, but there are still certain isolated ranges of r (islands of stability) that show non-chaotic behavior.

This is a rich landscape of possibilities to explore. For instance, with $r=3.2$ we get a period-2 cycle, oscillating between $x_\pm$ with values

```
\eval[sep=\ \text{and}\ ,p=.,ff]
    { (1/2r)[r+1+\sqrt{(r-3)(r+1)}] ,
      (1/2r)[r+1-\sqrt{(r-3)(r+1)}] }[r=3.2]
```

$\Longrightarrow$ 0.799455 and 0.513045.

`\iter[do=18,see=6]{\[ rx(1-x) \]}[r=3.2,x=0.5]` $\Longrightarrow$

$$rx(1-x)=0.8,\qquad (r=3.2,x=0.5)$$

$\ldots$ final 6 of 18:
$\hookrightarrow$ 0.799455
$\hookrightarrow$ 0.513044
$\hookrightarrow$ 0.799455
$\hookrightarrow$ 0.513045
$\hookrightarrow$ 0.799455
$\hookrightarrow$ 0.513045

With $r = 3.6$ we get chaos. For initial value of x I have chosen neighbouring values, 0.3 and 0.31:

`\iter[env=\[,do=100,see=6]{ rx(1-x) }[r=3.6,x=0.3]` $\Longrightarrow$

$$rx(1-x)=0.756,\qquad (r=3.6,x=0.3)$$

$\ldots$ final 6 of 100:
$\hookrightarrow$ 0.849022
$\hookrightarrow$ 0.46146
$\hookrightarrow$ 0.894653
$\hookrightarrow$ 0.339297
$\hookrightarrow$ 0.807028
$\hookrightarrow$ 0.560641

`\iter[env=\[,do=100,see=6]{ rx(1-x) }[r=3.6,x=0.31]` $\Longrightarrow$

$$rx(1-x)=0.77004,\qquad (r=3.6,x=0.31)$$

$\ldots$ final 6 of 100:
$\hookrightarrow$ 0.864724
$\hookrightarrow$ 0.421115
$\hookrightarrow$ 0.877598
$\hookrightarrow$ 0.386712
$\hookrightarrow$ 0.853797
$\hookrightarrow$ 0.44938

If you are using the decimal comma, make sure that the variables in the vv-list are separated by *semicolons* rather than commas, otherwise puzzling errors are almost certain to arise.

2.2 Star (*) option: fixed points

In some of the preceding examples, iteration eventually ended at a *fixed point* – a point x where $f(x) = x$. Appending a star (asterisk) to the `\iter` command is the signal for the `\iter` command to continue iterating until a fixed point is reached at the specified rounding value, or some fixed maximum number of iterations have been performed. The star overrides any value specified by the the `do` setting. It also overrides any elements of the display other than the numerical result – meaning negative results display with a hyphen for the minus sign unless `\iter*` is placed in a math environment.

Return to an earlier example, the continued fraction $1 + 1/x$. Starring the `\iter` command gives

`\iter*{ 1+1/x }[x=1]` $\Longrightarrow$ 1.618034,

and generalizing the example,

`\iter*{ 1+a/x }[a=n(n+1),n=3,x=1]` $\Longrightarrow$ 4

Indeed, trying in turn $n = 0, 1, 2, 3, 4, 5$ we see that when iterated $1+a/x \to n+1$. A little invesigating shows that this is hardly surprising. If x is a point such that

$$1 + \frac{n(n+1)}{x} = x$$

then $x^2 - x - n(n+1) = 0$. The quadratic factorizes: $(x - (n+1))(x+n) = 0$ so that, indeed, $x = n+1$ is a fixed point, as also – we learn – is $x = -n$, trivially. There is nothing here that requires n to be an integer,

`\iter*{ 1+a/x }[a=n(n+1),n=(\surd5-1)/2,x=1]` $\Longrightarrow$ 1.618034,

but if we put $n = 6$, we get a message:

`\iter*{ 1+a/x }[a=n(n+1),n=6,x=1]` $\Longrightarrow$!!! No fixed point attained after 100 iterations of: formula. !!!

It is easy to increase the 100 here to a larger value as we will do in a moment but it is worth using the `\info` command from `numerica` to see what is happening. For $n = 1$ and initial value $x = 1$,

```
\iter*{1+a/x}[a=n(n+1),n=1,x=1],
we see that it takes \info{iter}
```

$\Longrightarrow$ 2, we see that it takes 23 iterations to attain a 6-figure fixed point; when $n = 2$ for this same initial value it takes 41; … ; and when $n = 5$ it takes 96. The message when $n = 6$ is hardly surprising.

The maximum prevents `\iter` falling into an infinite loop or similar state. We saw with the logistic map that there are parameter values that lead to 2-cycles, 4-cycles, 8-cycles … chaos, where iteration would continue indefinitely if there were no safeguard like a specified maximum number of iterations. For our case, however, it doesn't look as if infinite loops of this kind are the problem. We increase the maximum by using the setting `max`:

`\iter*[max=150]{1+a/x}[a=n(n+1),n=6,x=1], taking \info{iter}` $\Longrightarrow$ 7, taking 114 iterations,

and the fixed point is safely attained well within the bound of the new maximum.

Alternatively, we could have reduced the rounding value, say from the default 6 to 5:

`\iter*{1+a/x}[a=n(n+1),n=6,x=1][5], taking \info{iter}` $\Longrightarrow$ 7, taking 99 iterations.

A fixed point is attained – but with no room to spare. Generally, reducing the rounding value is the other strategy to pursue when faced with the 'No fixed point attained' message, and perhaps the better one initially. If there is no fixed point after 100 iterations at some low rounding value – say 2 or 3 – then there may well be no fixed point at all.

`\iter` determines that a fixed point has been attained when the difference between successive iterations vanishes when rounded to the current rounding value. This can lead to an error in the final digit: the *difference* may vanish but the final value round *away* from the fixed point. To seek reassurance that the fixed point really is the correct value you might wish to seek a fixed point at a higher rounding value without changing the number of digits displayed. The extra rounding is achieved by entering `+=<integer>` in the settings option, where `<integer>` is the *extra* rounding desired.

For example, for the logistics map with $r = 1.5$ we expect a fixed point with value

`\eval{1-1/r}[r=1.5]` $\Longrightarrow$ 0.333333,

and indeed

`\iter*{rx(1-x)}[r=1.5,x=0.5]` $\Longrightarrow$ 0.333334,

the expected value – if we ignore the final digit. So let's use the `+` setting, to demand that the difference between successive iterate values at $6+1 = 7$ figures vanishes. That should ensure the correct 6-figure fixed point is attained, and it is:

`\iter*[+=1]{rx(1-x)}[r=1.5,x=0.5]` $\Longrightarrow$ 0.333333.

2.2.1 Use with `\nmcInfo`

We have already seen that the `\nmcInfo` command provides information on the number of iterations necessary to attain a fixed point, especially how many are required at a particular rounding value. That knowledge allows a good guess as to whether a fixed point will be attained at a greater rounding value. Thus when iterating the function

$$f(t_{ij}) = c^{-1}\sqrt{r_i^2 + r_j^2 - 2r_ir_j\cos(\theta_j - \theta_i + \omega t_{ij})}$$

in §1.1 only 5 iterations were required to attain 6-figure accuracy for the fixed point. That information came by following the `\iter*` command with `\nmcInfo` (or `\info`) with the argument `iter`. And generally, for any 'infinite' process, follow the command with an `\info` command if you want to know how many 'steps' – in the present case iterations – are required to achieve the result. So, if 5 iterations achieve 6-figure accuracy, presumably something like 10 iterations will achieve 12-figure accuracy:

```
\iter*{ c^{-1}\sqrt{r_i^2+r_j^2-2r_i r_j
  \cos(\theta_{ij}+\omega t)}
    }[ r_i=10,r_j=20,\theta_{ij}=0.2,t=1 ][12]
,\quad\info{iter}.
```

$\Longrightarrow$ 0.356899026113 , 9 iterations. (Remember, `numerica-plus` knows the values of c and ω from a `\constants` statement in the preamble.) And indeed only 9 iterations suffice to achieve 12-figure accuracy:

```
\iter[env=\[,vv=,do =11,see=4]
  { c^{-1}\sqrt{r_i^2+r_j^2-2r_i r_j
    \cos(\theta_{ij}+\omega t)}
  }[ r_i=10,r_j=20,\theta_{ij}=0.2,t=1 ][12]
```

$\Longrightarrow$

$$\begin{aligned} c^{-1}\sqrt{r_i^2+r_j^2-2r_ir_j\cos(\theta_{ij}+\omega t)} &= 0.382354696292 \\ &\ldots \text{ final 4 of 11:} \\ &\hookrightarrow 0.356899026114 \\ &\hookrightarrow 0.356899026113 \\ &\hookrightarrow 0.356899026113 \\ &\hookrightarrow 0.356899026113 \end{aligned}$$

(Display of the vv-list has been suppressed with the setting `vv=` .)

Or again, with another example from earlier,

`\iter*{\cos x}[x=\pi/2],\ \info{iter}` $\Longrightarrow$ 0.739085, 37 iterations.

That suggests that around $2 \times 37 = 74$ iterations will give a $2 \times 6 = 12$-figure answer, well within the cut-off figure of 100:

`\iter*{\cos x}[x=\pi/2][12],\ \info{iter}.` $\Longrightarrow$ 0.739085133215, 72 iterations.

2.3 Settings, saving results, errors,

The settings option is a comma-separated list of items of the form *key = value*. Only some of the keys for `\nmcEvaluate` discussed in Chapter 5 of the associated document `numerica.pdf` are relevant for `\nmcIterate`. Thus should a quantity in the vv-list depend on the iteration variable, forcing an implicit mode calculation, simply enter `vv@=1` (alternatively, `vvmode=1`) in the settings option, as with `\eval`:

`\iter*[vv@=1]{ f(x) }[f(x)=1+a/x,a=12,x=1]` $\Longrightarrow$ 4.

Implicit in the example is the default multi-token setting `xx=1` inherited from `\eval` and ensuring that the multi-token variable $f(x)$ is treated correctly.

We could add `dbg=1` to the example – or just enter `view` – to get a glimpse at the ‘innards’ of what is going on:

```
\iter*[view,vv@=1]{ f(x) }[f(x)=1+a/x,a=12,x=1]
\info{iter}
```

⟹

function: \nmc_x
vv-list: \nmc_x =1+a/x, a=12, x=1
stored: \nmc_x =3.999999832569298, a=12, x=4.000000223240948
fp-form: (3.999999832569298)
LaTeX: 4

59 iterations

The multi-token variable `f(x)` has been changed to a single-token. The difference between the two long 'stored' numbers is less than 5 in the 7th decimal place, meaning a fixed point has been found. Since 59 iterations are required to attain the fixed point at 6-decimal place accuracy, the values shown correspond to the 60th iteration, the *final* iteration. Because the `\iter` command is the starred form, the result that is fed to LaTeX is simply the fixed point 4 expressed as a number. Remove the star, add `do=60` and replace `view` with `dbg=55` (or equivalently `dbg=5*11`) in the settings option:

`\iter[dbg=55,vv@=1,do=60]{ f(x) }[f(x)=1+a/x,a=12,x=1]` ⟹

stored: \nmc_y =3.999999832569298, a=12, x=4.000000223240948
LaTeX: $\begin {array}{r@{}l}&{}13,\mskip 12muplus6muminus9mu(f(x)=1+a/x,a=12,x=1)\\&\ldots \ \mbox {final\ 4\ of\ 60:}\\&\hookrightarrow 4\\&\hookrightarrow 4\\&\hookrightarrow 4\\&\hookrightarrow 4\end {array}$

Now the LaTeX form is much fuller and the stored numbers exactly match those of the starred form.

2.3.1 \nmcIterate-specific settings

In addition to the inherited settings there are some specific to `\nmcIterate`. These are listed in Table 2.1.

2.3.1.1 Iteration variable

In nearly all of the examples so far, the iteration variable has been the rightmost variable in the vv-list and has not needed to be otherwise specified. However it is sometimes not feasible to indicate the variable in this way. In that case, entering

```
var = <variable name>
```

Table 2.1: Settings for `\nmcIterate`

key	type	meaning	initial
`var`	token(s)	iteration variable	
`+`	int	fixed point extra rounding	`0`
`max`	int > 0	max. iteration count (fixed points)	`100`
`do`	int > 0	number of iterations to perform	`5`
`see`	int > 0	number of final iterations to view	`4`
	int (`0/1/2`)	form of result saved with `\`	`0`

in the settings option enables the variable to be specified, irrespective of what the rightmost variable in the vv-list is. Here, `<variable name>` will generally be a character like `x` or `t` or a token like `\alpha`, but it could also be a multi-token name like `x'` or `\beta_{ij}` (or even `Fred` if you so chose). Although the iteration variable can be independently specified like this, it must still be given an initial *value* in the vv-list – only now it need not be the rightmost variable.

In the following example the rightmost variable is `n` which is clearly *not* the iteration variable:

`\iter[var=x,do=40]{$ 1+a/x $}[x=n-1,a=n(n+1),n=2][*]` $\Longrightarrow$

$$1+a/x=7.000000,\quad(x=n-1,a=n(n+1),n=2)$$

$\ldots$ final 4 of 40:
$\hookrightarrow 3.000001$
$\hookrightarrow 2.999999$
$\hookrightarrow 3.000000$
$\hookrightarrow 3.000000$

2.3.1.2 Extra rounding for fixed-point calculations

The criterion used to signal the attainment of a fixed point is that the difference between successive iterations vanishes when rounded to the current rounding value. As already noted, because of rounding errors and the 'round to even' tie-breaking rule, this criterion may lead to error in the last digit. That can be avoided by rounding to a greater number of digits than the actual rounding value. This *extra* rounding is achieved by entering

```
+ = <integer>
```

in the settings option. By default this extra rounding is set to zero.

We have seen before that $\cos x$ starting at $x=\frac{1}{2}\pi$ takes 37 iterations to reach a 6-figure fixed point 0.739085, about 6 iterations per decimal place. By entering `+=1` in the settings option the number of iterations is increased to 43, 6 more than 37 but, reassuringly, the 6-figure result that is displayed remains unchanged:

`$ \iter*[+=1]{\cos x}[x=\pi/2] $,\ \info{iter}` $\Longrightarrow$ 0.739085, 43 iterations.

2.3.1.3 Maximum iteration count for fixed-point searches

To prevent a fixed-point search from continuing indefinitely, perhaps because no fixed point exists, there needs to be a maximum number of iterations specified after which point the search is called off. By default this number is 100. To change it enter

```
max = <positive integer>
```

in the settings option.

2.3.1.4 Number of iterations to perform

To specify the number of iterations to perform enter

```
do = <positive integer>
```

in the settings option. Note that if the `*` option is present this value will be ignored and iteration will continue until either a fixed point or the maximum iteration count is reached. By default `do` is set to 5. (Note that `do` can be set to a greater number than the initial 100 setting of `max`; `max` applies only to the starred form `\iter*`.)

2.3.1.5 Number of iterations to show

To specify the number of final iterations to show enter

```
see = <positive integer>
```

in the settings option. By default `see` is set to 4. Always it is the *last* `see` iterations that are displayed. If `see` is set to an equal or greater value than `do`, all iterations are shown. If the star option is used the `see` value is ignored.

2.3.2 Form of result saved by `\nmcReuse`

In version 2 of `numerica-plus` there was a setting `reuse` that determined the content of what was saved with the `\nmcReuse` command. This has been removed. Now it is only the last `see` values that are saved as a comma list. The values stored in the list are each wrapped in braces. In this way no confusion arises if the decimal comma is being used. For a fixed point calculation, it is the fixed point as displayed that is saved.

```
\iter[do=12,see=4]
  {\[ kx(1-x) \]}[k=3.5,x=0.5]
\reuse{logistic}
```

$\Longrightarrow$

$$kx(1-x)=0.875,\qquad (k=3.5,x=0.5)$$

$\ldots$ final 4 of 12:
$\hookrightarrow 0.874997$
$\hookrightarrow 0.38282$
$\hookrightarrow 0.826941$
$\hookrightarrow 0.500884$

whence `$\logistic$` $\Longrightarrow$ $0.874997, 0.38282, 0.826941, 0.500884$. (I have placed `\logistic` between `$` delimiters to get thin spaces after the commas.)

If you want to isolate just one member of the comma list, `numerica-plus` offers the utility function `\clitem` ('comma list item') which acts on the control sequence followed by a number:

`\clitem\logistic 3` $\Longrightarrow$ 0.826941.

Use positive numbers for counting from the left, negative numbers for counting from the right. Multi-digit numbers must be enclosed in braces. See §4.3.2.1 for more on the use of `\clitem`.

2.3.3 Errors

There is only one error specifically related to `\nmcIterate` and that is when the number of iterations exceeds the specified maximum number in a fixed point calculation. We have already met this in the earlier discussion of fixed points. Another example is

`\iter*{kx(1-x)}[k=3.5,x=0.5]` $\Longrightarrow$!!! No fixed point attained after 100 iterations of: formula. !!!

Other `numerica` errors can also afflict an iteration calculation. Again an example of this was provided in §2.1 when we sought to iterate `\arccos` 38 times rather than the previously successful 37:

`\iter[do=38,see=4]{\[ \arccos x \]}[x=0.739085]` $\Longrightarrow$!!! `l3fp` error 'Invalid operation' in: formula. !!!

The 38th iterate is attempting to take the inverse cosine of a number greater than 1; `numerica` objects.

2.4 Newton-Raphson method

This is a method for solving equations in one variable, say $f(x)=0$, by iterating the expression

$$x_{n+1}-\frac{f(x_n)}{f'(x_n)}$$

where f' is the derivative of f. `numerica-plus` does not automatically calculate the derivative; given the function f the user needs to manually insert (the analytic expression for) it.

Fig. 2.1 depicts a situation where the method works well. This will not always be the case. If $x = z$ is the zero ($f(z) = 0$) then should the derivative vanish at z ($f'(z) = 0$) there may well be problems. I give an example below, after first giving a simple illustration of the method.

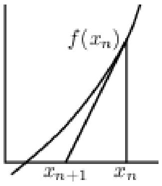

Figure 2.1: N-R method

Suppose $f(x) = \sin x$; then $f'(x) = \cos x$. If the initial test value is 4, then $x - \sin x/\cos x = x - \tan x$ is the expression to insert in the `\iter*` command:

`\iter*{x-\tan x}[x=4], \info{iter}` $\Longrightarrow$ 3.141593, 3 iterations,

which we recognize as the 6-figure value of π, attained very quickly after only 3 iterations. To check this omit the asterisk:

`\iter[f=1]{x-\tan x}[x=4]` $\Longrightarrow$

$$\begin{aligned} x - \tan x &= 2.842179, \quad (x = 4) \\ &\hookrightarrow 3.150873 \\ &\hookrightarrow 3.141592 \\ &\hookrightarrow 3.141593 \\ &\hookrightarrow 3.141593 \end{aligned}$$

But that is 4 iterations, not the claimed 3, before the correct value appears. This is an example of the difference between successive values vanishing at the 6-figure rounding value, but the final digit being 1 shy of the correct value. If we put `+=1` in the settings and round to 7 figures, we get

`\iter[f=1,+=1]{x-\tan x}[x=4][7]` $\Longrightarrow$

$$\begin{aligned} x - \tan x &= 2.8421787, \quad (x = 4) \\ &\hookrightarrow 3.1508729 \\ &\hookrightarrow 3.1415924 \\ &\hookrightarrow 3.1415927 \\ &\hookrightarrow 3.1415927 \end{aligned}$$

The difference between the 3rd and 4th values is 0.0000003 which rounds to 0 at 6 figures whereas 3.1415924 rounds to 3.141592.

We could also do the calculation in implicit mode (the `f=1` in the settings means the formula is part of the display and has *nothing* to do with the `f` in the main argument and vv-list):

`\iter[f=1,vv@=1]{x-f(x)/f'(x)}[f(x)=\sin x,{f'(x)}=\cos x,x=4]` $\Longrightarrow$

$$\begin{aligned} x - f(x)/f'(x) &= 2.842179, \quad (f(x) = \sin x, x = 4) \\ &\hookrightarrow 3.150873 \\ &\hookrightarrow 3.141592 \\ &\hookrightarrow 3.141593 \\ &\hookrightarrow 3.141593 \end{aligned}$$

where display of the derivative from the vv-list has been suppressed (by the enclosing braces).

Let's tackle something less obvious, finding the roots of $\cos x \cosh x \pm 1$. (*HMF* Table 4.18 gives small tables of the first five roots in both cases; subsequent roots are essentially $\cos^{-1}(\mp 1) = \frac{1}{2}(2n \pm 1)\pi$ since $\cosh x$ is so large.) Writing $f(x)$ for the function, in both cases $f'(x) = \cos x \sinh x - \sin x \cosh x$. In the example, our initial test value is `x=5`. We save the result in the control sequence `\zilch` and then check that it really does contain a zero of $f(x)$ in the final statement:

```
\iter*[vv@=1]{ x-f(x)/f'(x) }
  [f(x)=\cos x\cosh x-1, {f'(x)}=
    \cos x\sinh x-\sin x\cosh x,x=5][15],
\quad \info{iter}
\reuse[renew]{zilch} \macros{ \zilch } \par
\eval{\[\cos x\cosh x -1 \]}[x=\zilch][15]
```

$\Longrightarrow$ 4.730040744862704, 5 iterations

$$\cos x \cosh x - 1 = -0.000000000000001, \qquad (x = 4.730040744862704)$$

It is remarkable that only 5 iterations are required for 15-place accuracy; *HMF* gives only 6-figures, 4.7300407.

Of course, $x = 0$ is a zero of $\cos x \cosh x - 1$, but $f'(x)$ vanishes at zero and the Newton-Raphson method fails there for that reason. Indeed it's easy to check that $f'(0) = f''(0) = f'''(0) = 0$ and $f^{iv}(x) = -4(f(x) + 1)$ meaning $f^{iv}(0) = -4$ and on repeated differentiation the pattern repeats. Thus the Maclaurin series for $f(x)$ near $x = 0$ is

$$f(x) = \sum_{n=1}^{\infty} \frac{(-1)^n 4^n x^{4n}}{(4n)!} = -\frac{x^4}{6} + \frac{x^8}{2520} - \dots$$

Any value of x close to 0 is going to give a value of $f(x) = \cos x \cosh x - 1$ about 4 decimal places closer. Indeed, when trial values like $x = 1$, $x = 0.5$, $x = 0.3$ and so on are used in the iteration of $x - f(x)/f'(x)$, the iteration converges on a series of 'pseudo-zeros' of $f(x)$, all artifacts of the rapidity with which $\cos x \cosh x$ converges to 1 at $x = 0$.

Chapter 3

Finding zeros and extrema: `\nmcSolve`

`numerica-plus` provides a command, `\nmcSolve` (short-name form `\solve`), for finding a zero of a function, should it have one. In the following example,

`\solve[p]{\[ e^{ax}-bx^2 \]}[a=2,b=3,{x}=0]` $\Longrightarrow$

$$e^{ax}-bx^2=-0.000002,\qquad(a=2,b=3)\;\rightarrow\;x=-0.390647,$$

I have sought and found a solution x to the equation $e^{ax/2}-bx^2=0$ when $a=2$ and $b=3$, starting with a trial value $x=0$, entered as the *rightmost* variable in the vv-list (and em-braced since I don't want this trial value displaying in the presentation of the result). Although x has been found to the default six-figure accuracy, it is evident that the function vanishes only to five figures. Let's check:

`\eval{$ bx^2 $}[b=3,x=x=-0.390647]` $\Longrightarrow$
$bx^2=0.457815,\quad(b=3,x=-0.390647),$
`\eval{$ e^{ax} $}[a=2,x=-0.390647]` $\Longrightarrow$
$e^{ax}=0.457813,\quad(a=2,x=-0.390647);$

the values agree save in the final digit.

This discrepancy in the final decimal place or places is a general feature of solutions found by `\solve`. It is the value of x, not the value of $f(x)$, that is being found (in this case) to six figures. If the graph of a function crosses the x-axis steeply then the x value (the zero) may be located to a higher precision than the function value. Conversely, if the graph of a function crosses the x-axis gently (at a shallow angle) then the function value will vanish to a greater number of decimal places than the zero (the x value) is found to.

A second example, which we can check against values tabulated in *HMF*, is to find a value of x that satisfies $\tan x=\lambda x$. In other words, find a zero of $\tan x-\lambda x$. In the example λ is negative, so a trial value for x greater than $\pi/2$ seems like a good idea. I've chosen $x=2$.

`\solve{$ \tan x - \lambda x $}[\lambda=-1/0.8,{x}=2][5]` $\Longrightarrow$
$\tan x - \lambda x = -0.00002, \quad (\lambda = -1/0.8) \;\rightarrow\; x = 1.95857.$

Table 4.19 of *HMF* lists values of x against λ and this is the 5-decimal place value tabulated there.

3.1 Extrema

A function may not have a zero; or, for the given initial trial value and initial step in the search for a zero, there may be a local extremum in the way. In that case `numerica-plus` may well locate the local extremum (maximum or minimum but not a saddle point). For example for the quadratic $(2x-1)^2+3x+1$ the `\solve` command gives the result

`\solve[vv=]{$ (2x-1)^2+3x+1 $}[x=2]` $\Longrightarrow$
$(2x-1)^2+3x+1 = 1.9375 \;\rightarrow\; x = 0.124999.$

Since $(2x-1)^2+3x+1 \neq 0$ for any (real number) x, we deduce that the quadratic takes a minimum value 1.9375 at $x = 0.125$ – easily confirmed analytically. This particular minimum is a global minimum but in general any extremum found is only *local*. The function may well take larger or smaller values (or vanish for that matter) further afield.

It is also worth noting in this example the `vv=` in the settings option which suppresses display of the vv-list. (The only member of the vv-list is the trial value `x=2` which we do not want to display.)

Note that the function for which a zero is being sought is *not* equated to zero when entered in the `\solve` command. It is `\solve{ f(x) }`, not `\solve{ f(x)=0 }`. This is precisely because it may be an extremum that is found rather than a zero (if extremum or zero is found at all – think e^x). The display of the result makes clear which is which, equating $f(x)$ to its value, zero or extremum depending on what has been found, as you can see in the preceding examples.

3.1.1 The search strategy

If you have some sense of where a function has a zero, then choose a trial value in that vicinity. `\solve` uses a bisection method to home in on the zero. It therefore needs *two* initial values. For the first it uses the trial value you specify, call it a and for the second, by default, it uses $a+1$. (The default value 1 for the initial step from the trial value can be changed in the settings option with the setting `dvar`; see §3.3.) If $f(a)$ and $f(a+1)$ have opposite signs then that is good. Bisection of the interval $[a, a+1]$ can begin immediately in order to home in on the precise point where f vanishes. Write $b = a+1$.

- Let $c=\frac{1}{2}(a+b)$; if $f(c)=0$ the zero is found; otherwise either $f(a),f(c)$ are of opposite signs or $f(c),f(b)$ are of opposite signs. In the former case write $a_1=a$, $b_1=c$; in the latter case write $a_1=c$, $b_1=b$ and then redefine $c=\frac{1}{2}(a_1+b_1)$. Continue the bisection process, either until an exact zero c of f is reached ($f(c)=0$) or a value c is reached where the difference between a_{n+1} and b_{n+1} is zero at the specified rounding value. (But note, $f(c)$ may not vanish at that rounding value – the zero might be elsewhere in the interval and f might cross the axis at a steep slope.)

However $f(a)$ and $f(b)=f(a+1)$ may not have opposite signs. If we graph the function $y=f(x)$ and suppose $f(a),f(b)$ are distinct but of the same sign, then the line through the points $(a,f(a))$, $(b,f(b))$ will intersect the x-axis to the left of a or the right of b depending on its slope. We search always *towards the x-axis* in steps of $b-a$ ($=1$ with default values).

- If the line intersects the axis to the left of a then $c=a-(b-a)$ and we set $a_1=c,b_1=a$; if the line intersects the axis to the right of b then $c=b+(b-a)$ and we set $b_1=c,a_1=b$. The hope is that by always taking steps in the direction towards the x-axis that eventually $f(c)$ will be found to lie on the *opposite* side of the axis from $f(a_n)$ or $f(b_n)$, at which point the bisection process begins.

- Of course this may not happen. At some point c may lie to the left of a_n but $|f(c)|>|f(a_n)|$, or c may lie to the right of b_n but $|f(c)|>|f(b_n)|$. The slope has reversed. In that case we halve the step value to $\frac{1}{2}(b-a)$ and try again in the same direction as before from the same point as before (a_n or b_n as the case may be).

- Should we find at some point that $f(a_n)=f(b_n)$ then the previous strategy does not apply. In this case we choose a_{n+1} and b_{n+1} at the quarter and three-quarter marks between a_n and b_n. Either $f(a_{n+1})$ and $f(b_{n+1})$ will differ and the previous search strategy can start again or we are on the way to finding an extremum of f.

As already noted it is also possible that our function has neither zeros nor extrema. To prevent the search continuing indefinitely, `numerica` uses a cut-off value for the maximum number of steps pursued – by default set at 100.

3.1.2 Elusive extrema

The strategy ‘search always towards the x-axis’ has a consequence: it means that a local maximum above the x-axis will almost certainly not be found, since ‘towards the x-axis’ pulls the search away from the maximum. Similarly a local minimum below the x-axis will also not be found since ‘towards the x-axis’ pulls the search away from the minimum.

One way of countering this elusiveness is to add a constant value (possibly negative) to the function whose zeros and extrema are being sought. The zeros of the function will change but the abscissae (x values) of the extrema remain unchanged. If the constant is big enough it will push a local minimum above the axis where it can be found or, for a negative constant, push a local maximum below the axis where it can be found.

For example $f(x)=x^3-x$ has roots at $-1,0,1$, a local maximum at $-\frac{1}{\sqrt{3}}$ and a local minimum at $\frac{1}{\sqrt{3}}$. To locate the minimum, I have added an unnecessarily large constant k to $f(x)$ ($k=1$ would have sufficed) and a start value at the rightmost zero. Searching 'towards the x-axis' will take the search towards the local minimum.

`\solve{$ x^3-x+k $}[k=5,{x}=1]` $\Longrightarrow$

$$x^3-x+k=4.6151,\quad (k=5)\ \to\ x=0.577351.$$

Checking, `\eval{$\tfrac1{\surd 3}$}` $\Longrightarrow$ $\frac{1}{\sqrt{3}}=0.57735$. There is a discrepancy in the 6th decimal place which can be eliminated by using the extra rounding setting; see §3.3.1.3. The value of the local minimum of x^3-x is then $4.6151-5=-0.3849$.

Or, we can find the value of that local minimum value and the x value where it occurs 'in one go' by *nesting* `\solve` within the vv-list of an `\eval` command:

`\eval{$ x^3-x $}[x={\solve{y^3-y+k}[k=5,y=1]}]` $\Longrightarrow$

$$x^3-x=-0.3849,\quad (x=0.577351).$$

The braces around the `\solve` and its vv-list hide *its* square-brackets from the parsing of the vv-list of the `\eval` command.

3.1.3 False extrema

A function which 'has an infinity' at a particular value can result in a false extremum being found:

`\solve{$ 1/x $}[x=-1/3]` $\Longrightarrow$

$$1/x=-3145728.00033,\quad (x=-1/3)\ \to\ x=0.$$

One needs to look for extrema with some awareness of how the function behaves. 'Searching blind' may lead to nonsense results. In this particular example, changing the rounding value will show the supposed extremum jumping from one large value to another and not settling at any particular value.

3.2 Star (*) option

A starred form of the `\nmcSolve` command gives a purely numerical result; should it be negative it displays with a hyphen for the minus sign outside a math environment. All other elements of the display are suppressed. When commands are nested, `numerica` and associated packages treat an inner command as if it were the starred form, whether the star is explicitly present or not. Returning to a previous example,

`\solve*{$ \tan x - \lambda x $}[\lambda=-1/0.8,{x}=2][5]` $\Longrightarrow$
1.95857,

giving the zero and nothing more.

3.3 Settings option

The settings option is a comma-separated key-value list of items. The keys discussed in the settings option for `\nmcEvaluate` discussed in the associated document `numerica.pdf` are also available for `\nmcSolve`. The very first example in this chapter used the punctuation option `p` (`\solve[p]{\[...`) to ensure a comma after the display-style presentation of the result. We also saw in the quadratic example illustrating extrema the use of `vv=` with no value to suppress display of the vv-list: `\solve[vv=]{$ ...`.

Putting `dbg=1`, or more simply, entering `view` in the settings option produces a familiar kind of display. Using the function

$$ct-\sqrt{a^2+b^2-2ab\cos(\beta+\omega t)}$$

from the rotating disk problem,

```
\solve[view,var=t]
  {$ ct-\sqrt{a^{2}+b^{2}-2ab\cos(\beta+\omega t)} $}
    [a=10,b=20,\beta=1,{t}=0][4]
```

$\Longrightarrow$

```
function: ct-\sqrt {a^{2}+b^{2}-2ab\cos (\beta +\omega t)}
 vv-list: a=10, b=20, \beta =1, t=0
  stored: a=10, b=20, \beta =1, t=0.601715087890625
 fp-form: (30)(0.601715087890625)-sqrt((10)^(2)+(20)^(2)-
          2(10)(20)cos(((1)+(0.2)(0.601715087890625))))
   LaTeX: $ct-\sqrt {a^{2}+b^{2}-2ab\cos (\beta +\omega t)}=-0.0007,\mskip
          12muplus6muminus9mu(a=10,b=20,\beta =1)\protect \protect
          \leavevmode@ifvmode \kern +.1667em\relax \to \protect \protect
          \leavevmode@ifvmode \kern +.1667em\relax t=0.6017$
```

If that is too much information, understand that `view` (or `dbg=1`) for `\nmcSolve` is equivalent to `dbg=2*3*5*7*11`. Entering `dbg=2*3*5*7` for instance will trim the LaTeX expression from the debug display, and similarly, omitting other prime factors from `dbg` will result in the corresponding element being absent from the display. (The prime factors can be multiplied out if you wish; see Chapter 5 of the associated document `numerica.pdf`.)

3.3.0.1 Multi-line display of the result

By default the result is presented on a single line (unless the star option is being used). This can be of the form *function = function value, (vv-list) → variable = result*, where *function value* will either be 0 or close to it, or an extremum of the function, and *result* will be the value of the *variable* producing that zero or extremum when substituted into the function. It takes only a slightly complicated formula and only a few variables in the vv-list before this becomes an overcrowded line, exceeding the line width and extending into and perhaps beyond the margin. To split the display over two lines specify a `multline` or `multline*` environment in the settings option (if you don't want an equation number use the starred form):

```
\solve[env=multline*,p=.]
  { ct-\sqrt{a^{2}+b^{2}-2ab\cos(\beta+\omega t)}
  }[a=10,b=20,\beta=1,{t}=0][4]
```

⟹

$$\begin{multline*} ct-\sqrt{a^{2}+b^{2}-2ab\cos(\beta+\omega t)}=-0.0007,\\ (a=10,b=20,\beta=1)\ \rightarrow\ t=0.6017. \end{multline*}$$

For a formula with a significantly longer vv-list you could introduce a third line to display the arrow and result on by entering a specification like `vv={,}\\(vv)\\` in the settings option *after* the `env` setting.

In the example the function evaluates to −0.0007. Is this a zero or an extremum? To find out, the calculation needs to be carried out to a higher rounding value – which is the reason why `\nmcSolve` has an extra rounding setting; see §3.3.1.3 below.

3.3.1 `\nmcSolve`-specific settings

In addition to inherited settings there are some settings specific to `\nmcSolve`. These are listed in Table 3.1.

Table 3.1: Settings for `\nmcSolve`

key	type	meaning	initial
`var`	token(s)	equation variable	
`dvar`	real $\neq 0$	initial step size	`1`
`+`	int	extra rounding	`0`
`max`	int > 0	max. number of steps before cut off	`100`

3.3.1.1 Equation variable

By default the equation variable is the *rightmost* variable in the vv-list. This may not always be convenient. A different equation variable can be specified by entering

```
var = <variable name>
```

in the vv-list. `<variable name>` will generally be a single character or token but multi-token names are perfectly acceptable (with `numerica`'s default multi-token setting; see the associated document `numerica.pdf` about this).

3.3.1.2 Initial step size

The vv-list contains the equation variable set to the initial trial value. But `\solve` needs *two* initial values to begin its search for a zero or extremum; see §3.1.1. Ideally, these values will straddle a zero of the function being investigated. 'Out of the box', the second trial value is 1 more than the first: if the equation variable is set to trial value a then the second value defaults to $a+1$. The '+1' here can be changed by entering in the settings option

```
dvar = <non-zero real number>
```

For instance, `dvar=-1`, or `dvar=\pi` are two valid specifications of initial step size. The notation is prompted by the use of expressions like x and $x+dx$ for two nearby points in calculus.

The initial step value may be too big or too small. An example where the default step value is too big and a smaller one needs to be specified is provided by Planck's radiation function (*HMF* Table 27.2),

$$f(x)=\frac{1}{x^5(e^{1/x}-1)}.$$

From the (somewhat coarse-grained) table in *HMF* it is clear that there is a maximum of approximately 21.2 when x is a little more than 0.2. This is a maximum above the x-axis and hence 'elusive' in the sense of §3.1.2. To find it, substract 100 (say) from the formula and again use the ability to nest commands to display the result. In the example, I find in the vv-list of the `\eval` command the value of x which maximizes the Planck radiation function, then calculate the maximum in the main argument of the `\eval` command. Note the `dvar=0.1` in the settings option of the `\solve` command:

```
\eval[p=.]{\[ \frac1{x^5(e^{1/x}-1)} \]}
  [ x={ \solve[dvar=0.1]
    { \frac1{y^5(e^{1/y}-1)}-100 }[y=0.1] } ]
```

$\Longrightarrow$

$$\frac{1}{x^5(e^{1/x}-1)}=21.201436,\qquad(x=0.201405).$$

The maximum is indeed a little over 21.2 and the x value a little more than 0.2.

The default `dvar=1` is too big for this problem. From the table in *HMF*, $f(0.1) = 4.540$ and $f(1.1) = 0.419$. Thus for $f(x) - 100$ the 'towards the x-axis' search strategy would lead to negative values of x with the default `dvar` setting.

3.3.1.3 Extra rounding

`\solve` determines that a zero or an extremum has been reached when the difference between two successive bisection values vanishes at the specified rounding value (the value in the final trailing optional argument of the `\solve` command; 6 by default). If our function is $f(x)$ then $|x_{n+1} - x_n| = 0$ to the specified rounding value and $f(x_n)$, $f(x_{n+1})$ have opposite signs or at least one vanishes. Then (assuming $x_{n+1} > x_n$ and continuity) there must be a critical value $x_c \in [x_n, x_{n+1}]$ such that $f(x_c) = 0$ exactly. But in general the critical value x_c will not coincide with x_n or x_{n+1}. If $f(x)$ crosses the x-axis at a steep angle it may well be that although $f(x_c)$ vanishes to all 16 figures, $f(x_n)$ and $f(x_{n+1})$ do not, not even at the (generally smaller) specified rounding value. For instance, suppose $f(x) = 1000x - 3000$ and that our trial value is $x = e$:

`\solve[vv=]{$ 1000x-3000 $}[x=e][4*]` $\Longrightarrow$

$$1000x - 3000 = -0.0409 \;\rightarrow\; x = 3.0000.$$

Although the difference between successive x values vanishes to 4 places of decimals, $f(x)$ does not, not even to 2 places. If we want the function to vanish at the specified rounding value – 4 in the example – then we will need to locate the zero more precisely than that.

This is the purpose of the extra rounding key in the settings option. Enter

```
+ = <integer>
```

in the settings option of the `\solve` command to add `<integer>` to the general rounding value. By default, `+=0`.

With this option available it is easy to check that `+=3` suffices in the example to ensure that both x and $f(x)$ vanish to 4 places of decimals,

`\solve[+=3]{$ 1000x-3000 $}[x=e][4*]` $\Longrightarrow$

$$1000x - 3000 = 0.0000, \quad (x = e) \;\rightarrow\; x = 3.0000,$$

and that `+=2` does not, i.e., we need to locate the zero to $4 + 3 = 7$ figures to ensure the function vanishes to 4 figures.

There is no need for the `<integer>` to be positive. In fact negative values can illuminate what is going on. In the first of the following, the display is to 10 places but (`+=-4`) the calculation is only to $10 - 4 = 6$ places. In the second, the display is again to 10 places, but (`+=-3`) the calculation is to $10 - 3 = 7$ places.

`\solve[+=-4]{$ 1000x-3000 $}[x=e][10*]` $\Longrightarrow$

$$1000x - 3000 = -0.0008711259, \quad (x = e) \;\rightarrow\; x = 2.9999991289,$$

`\solve[+=-3]{$ 1000x-3000 $}[x=e][10*]` $\Longrightarrow$

$$1000x - 3000 = -0.0000366609, \quad (x = e) \;\rightarrow\; x = 2.9999999633.$$

Only in the second does $f(x) = 1000x - 3000$ vanish when rounded to 4 figures.

Returning to an earlier example (§3.3.0.1) in which it was not entirely clear whether a zero or an extremum had been found, we can now resolve the confusion. Use the extra rounding setting (and pad with zeros to emphasize the 4-figure display by adding an asterisk in the trailing optional argument):

```
\solve[env=multline*,p=.,+=2]
  { ct-\sqrt{a^{2}+b^{2}-2ab\cos(\beta+\omega t)} }
  [a=10,b=20,\beta=1,{t}=0][4*]
```

$\Longrightarrow$

$$ct - \sqrt{a^2 + b^2 - 2ab\cos(\beta + \omega t)} = 0.0000, \\ (a = 10, b = 20, \beta = 1) \rightarrow t = 0.6017.$$

3.3.1.4 Maximum number of steps before cut-off

Once two function values have been found of different sign, bisection is guaranteed to arrive at a result. The problem is the *search* for two such values. This may not terminate – think of a function like e^x which lacks both zeros and extrema. To prevent an infinite loop, `\solve` cuts off the search after 100 steps. This cut-off value can be changed for a calculation by entering

```
max = <positive integer>
```

in the settings option.

To illustrate, we know that $1/x$ has neither zero nor extremum, but we do not get an infinite loop – we get an error message if we attempt to 'solve' $1/x$:

`\solve{ 1/x }[x=1]` $\Longrightarrow$!!! No zero/extremum found after 100 steps for function: $1/x$. !!!

3.3.1.5 Form of result saved by `\nmcReuse`

`\nmcReuse` saves (only) the numerical result. Version 2 offered a setting `reuse` providing a choice of what was saved. That has been removed in version 3.

Chapter 4

Recurrence relations: `\nmcRecur`

One of the simplest recurrence relations is that determining the Fibonacci numbers, $f_{n+2} = f_{n+1} + f_n$, with initial values $f_0 = f_1 = 1$. The command `\nmcRecur`, short-name form `\recur`, allows calculation of the terms of this sequence:

```
$ \nmcRecur[do=8,see1=8,...]
  { f_{n+2}=f_{n+1}+f_{n} }[f_{1}=1,f_{0}=1] $
```

$\Longrightarrow 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots$

The recurrence relation is entered in the main argument (between braces), the initial values in the vv-list trailing the main argument, and the display specification is placed in the settings option: `do=8` terms to be calculated, all 8 to be viewed (`see1=8`), and the display to be concluded by an ellipsis to indicate that the sequence continues (those are three dots/periods/full stops in the settings option, not an ellipsis glyph).

A more complicated recurrence relation determines the Legendre polynomials:

$$(n+2)P_{n+2}(x) - (2n+3)xP_{n+1}(x) + (n+1)P_n(x) = 0.$$

For the purposes of `\recur` we need P_{n+2} expressed in terms of the lower order terms:

$$P_{n+2}(x) = \frac{1}{n+2}\left((2n+3)xP_{n+1}(x) - (n+1)P_n(x)\right).$$

It is this standard form – the term to be calculated on the left, equated to an expression involving a fixed number of lower-order terms on the right – that `numerica-plus` works with. For $P_0(x) = 1,\ P_1(x) = x$ and $x = 0.5$, the terms are calculated like this:

```
\recur[env=multline*,do=11,see1=4,see2=2,
  vv={,}\\(vv)\\] { P_{n+2}(x)=\frac{1}{n+2}
```

```
      \Bigl((2n+3)xP_{n+1}(x)-(n+1)P_{n}(x)\Bigr) }
    [P_{1}(x)=x,P_{0}(x)=1,x=0.5][7]
```

$\Longrightarrow$

$$\begin{multline*}P_{n+2}(x)=\frac{1}{n+2}\Bigl((2n+3)xP_{n+1}(x)-(n+1)P_{n}(x)\Bigr),\\ (P_{1}(x)=x,P_{0}(x)=1,x=0.5)\\ \to 1,\ 0.5,\ -0.125,\ -0.4375,\ \ldots,\ -0.2678986,\ -0.1882286\end{multline*}$$

where $P_9(0.5)=-0.2678986$ and $P_{10}(0.5)=-0.1882286$ are the last two displayed values (and to seven figures are the values listed in *HMF* Table 8.1).

The examples also illustrate a consistent response with other commands of the `numerica` suite to math environments. For the Fibonacci sequence the `\recur` command lay between `$` delimiters and display of the formula was suppressed. For the Legendre polynomials, `multline*` was specified in the settings option and the formula was included in the display. (‘Formula show’, ‘formula hide’ can also be set with the `f=1`, `f=0` settings.) Note also the specification of the vv-list in the second example, spreading the display over three lines, and the `see2` setting, specifying how many terms to display. The first example displays a concluding ellipsis; the second example doesn’t. That is the result of the presence of the `...` (three periods) setting in the first example.

4.1 Notational niceties

More than any of the other commands in `numerica` and associated packages, `\nmcRecur` depends on getting the notation into a standard form. At least at this stage (version 3.0.0) of `numerica-plus`

- the terms of the recurrence must be *subscripted*: f_n, $P_n(x)$ are examples;
- the recurrence relation is placed in the main (mandatory) argument of `\nmcRecur` in the form: *high-order term=function of consecutive lower-order terms*;
- the initial-value terms in the vv-list must occur left-to-right in the order *high* to *low* order;
- the recurrence variable changes by 1 between successive terms.

The example for Legendre polynomials in particular shows what is required. The Fibonacci example is simpler, since the recurrence variable does not occur independently in the recurrence relation as it does with the Legendre polynomials, although in both cases the recurrence variable is absent from the vv-list.

4.1.1 Recurrence variable in the vv-list

The recurrence variable is required in the vv-list only when an implicit mode calculation is undertaken. To that end write A and B for the coefficients $2n+3$ and $n+1$ respectively in the Legendre recurrence. A and B will now need entries in the vv-list which means the recurrence variable will need a value assigned to it there too, and we will need to add `vv@=1` (or `vvmode=1`) to the settings option.

```
\recur[env=multline*,vvmode=1,do=11,see1=4,see2=2,
         ...,vv={,}\\(vv)\\]
  { P_{n+2}(x)=\frac{1}{n+2}
    \Bigl(AxP_{n+1}(x)-BP_{n}(x)\Bigr) }
  [P_{1}(x)=x,P_{0}(x)=1,x=0.5,A=2n+3,B=n+1,n=0]
```

$\Longrightarrow$

$$P_{n+2}(x)=\frac{1}{n+2}\Bigl(AxP_{n+1}(x)-BP_n(x)\Bigr),$$
$$(P_1(x)=x,P_0(x)=1,x=0.5,A=2n+3,B=n+1,n=0)$$
$$\to 1,\ 0.5,\ -0.125,\ -0.4375,\ \ldots,\ -0.267899,\ -0.188229,\ \ldots$$

Since the vv-list is evaluated from the right, the left-to-right high-to-low ordering of the initial-value terms means the value of the lowest order term is read first. Although `numerica-plus` depends on this order of occurrence of the terms, they do not need to be *consecutive* as in the examples so far (although it is natural to enter them in this way). `numerica-plus` reads the value of the subscript of only the right-most term (the lowest order term), increments it by 1 when reading the next recurrence term to the left, and so on. The reading of the subscript of the lowest order term in the vv-list provides the initial value of the recurrence variable.

In the following example I have placed other items between $P_1(x)$ and $P_0(x)$ in the vv-list (but maintained their left-to-right order) and given the recurrence variable n a ridiculous initial value $\pi^2/12$. (Because of the order in which things get done 'behind the scenes', *some* value is necessary so that the n in '$B=n+1$' does not generate an 'unknown token' message.) The result is unchanged.

```
\recur[env=multline*,vvmode=1,do=11,see1=4,see2=2,
         ...,vv={,}\\(vv)\\]
  { P_{n+2}(x)=\frac{1}{n+2}
    \Bigl(AxP_{n+1}(x)-BP_{n}(x)\Bigr) }
  [A=2n+3,P_{1}(x)=x,B=n+1,n=\pi^2/12,P_{0}(x)=1,x=0.5]
```

$\Longrightarrow$

$$P_{n+2}(x)=\frac{1}{n+2}\Bigl(AxP_{n+1}(x)-BP_n(x)\Bigr),$$
$$(A=2n+3,P_1(x)=x,B=n+1,n=\pi^2/12,P_0(x)=1,x=0.5)$$
$$\to 1,\ 0.5,\ -0.125,\ -0.4375,\ \ldots,\ -0.267899,\ -0.188229,\ \ldots$$

4.1.2 Form of the recurrence relation

As noted earler, the form of the recurrence must be entered in the main argument in the form: *highest order term = function of consecutive lower order terms*. The number of lower order terms is the order of the recurrence. The Fibonacci and Legendre polynomial recurrences are both second order and presented in the form: *term* $n+2$ = *function of term* $n+1$ *and term* n. We could equally have done

```
\nmcRecur[p,do=8,see1=8,...]
  {$ f_{n}=f_{n-1}+f_{n-2} $}
    [f_{1}=1,f_{0}=1]
```

$\Longrightarrow f_n = f_{n-1} + f_{n-2},\ \ (f_1 = 1, f_0 = 1) \to 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots,$ where now the recurrence is of the form *term* n = *function of term* $n-1$ *and term* $n-2$, or (adjusting the coefficients as well as the recurrence terms),

```
\recur[env=multline*,p=.,do=10,see1=4,see2=2,
         vv={,}\\(vv)\\]
  { P_{n+1}(x)=\frac{1}{n+1}
    \Bigl((2n+1)xP_{n}(x)-nP_{n-1}(x)\Bigr) }
      [P_{2}(x)=-0.125,P_{1}(x)=x,x=0.5]
```

$\Longrightarrow$

$$P_{n+1}(x) = \frac{1}{n+1}\Bigl((2n+1)xP_n(x) - nP_{n-1}(x)\Bigr),$$
$$(P_2(x) = -0.125, P_1(x) = x, x = 0.5)$$
$$\to 0.5,\ -0.125,\ -0.4375,\ -0.289062,\ \ldots,\ -0.267899,\ -0.188229.$$

The recurrence here is of the form *term* $n+1$ = *function of term* n *and term* $n-1$. This last example has one further 'wrinkle'. I've made $P_1(x)$ the lowest order term and decreased the number of terms to calculate by 1 accordingly.

4.1.3 First order recurrences (iteration)

The recurrence relations for both the Fibonacci sequence and Legendre polynomials are second order. There is no reason why the recurrence should not be of third or higher order or, indeed, lower. A first order recurrence provides an alternative means of iterating functions. `\recur` therefore provides a means to display the results of an iteration in a different form from `\iter`.

Iterating $1 + a/x$ in this way, 16 terms gives the sequence

```
\recur[do=16,see1=0,see2=3,...]{$
  x_{n+1}=1+a/x_{n}
    $}[x_{0}=1,a=1]
```

$\Longrightarrow x_{n+1} = 1 + a/x_n,\ \ (x_0 = 1, a = 1) \to 1.618037,\ 1.618033,\ 1.618034,\ \ldots$ to be compared with the example near the start of Chapter 2. (*That* effected 15 iterations; *this* uses 16 terms because of the extra $x_0 = 1$ term.)

4.2 Star (*) option

When the star option is used with the `\nmcRecur` command, only a single term, the *last*, is presented as the result. Repeating the penultimate calculation, but with the star option produces

```
\recur*[do=10]{ P_{n+1}(x)=\frac{1}{n+1}
    \Bigl((2n+1)xP_{n}(x)-nP_{n-1}(x)\Bigr) }
      [P_{2}(x)=-0.125,P_{1}(x)=x,x=0.5]
```

⟹ -0.188229
With the exception of `do`, any settings are ignored in the display of the result. The star option produces a purely numerical answer without any trimmings.

This seems something of a waste of the star option since it gives much the same result as choosing `do=10,see1=0,see2=1` – not *exactly* the same, since math delimiters are involved here, but sufficiently similar to make me wonder if I should change the starred form to apply only to those recurrences which approach a limit. The starred form would then produce the limiting value as its result (like `\iter*`). This is a possible change for future versions of `numerica-plus` and should be borne in mind if using `\recur*`.

4.3 Settings

The settings option is a comma-separated list of items of the form *key* = *value*. Because recurrence terms are necessarily multi-token, the multi-token key is hard-coded in `\recur` to `xx=1`.

4.3.0.1 Multi-line formatting of result

When the `\recur` command wraps around math delimiters, the `vv` setting is available to split display of the result over two or more lines. With the enhanced treatment of environments in version 3 of `numerica`, some of which carries over to `numerica-plus`, it is possible to spread a display over two lines simply by specifying `env=multline*` (or `env=multline` if you want equation numbers). The default vv-list specification in this case is `vv={,}\\(vv)` which pushes the vv-list and sequence of calculated values to the second line. If two lines still give a crowded result, `vv={,}\\(vv)\\` pushes the vv-list, centred, to a second line and the sequence of values, right aligned, to a third. If you wanted only the sequence of calculated values on a second line with the vv-list on the same line as the formula then (for insance) `vv={,}\qquad(vv)\\` achieves this:

```
\nmcRecur[do=8,see1=8,...,vv={,}\qquad(vv)\\,*]
  {$ f_{n+2}=f_{n+1}+f_{n} $}
    [f_{1}=1,f_{0}=1]
```

$\Longrightarrow f_{n+2} = f_{n+1} + f_n, \qquad (f_1 = 1, f_0 = 1)$
$\to 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots$

4.3.1 `\nmcRecur`-specific settings

In addition to the inherited settings there are some specific to `\nmcRecur`. These are listed in Table 4.1.

4.3.1.1 Number of terms to calculate

By entering

```
do = <integer>
```

in the settings option you can specify how many terms of a recurrence to calculate. The default is set to 7 (largely to show a sufficient number of terms of the Fibonacci series to begin to be interesting). Note that `<integer>` will generally *not* correspond to the subscript on the last term calculated since that also depends on the value of the subscript of the lowest order term in the vv-list.

4.3.1.2 Number of terms to display

By entering

```
see1 = <integer1>, see2=<integer2>
```

in the settings option, you can specify how many initial terms of the recurrence and how many of the final terms calculated you want to view. If the sum of these settings is less than the `do` setting, then the terms are displayed with an intervening ellipsis. If the sum is greater than the `do` setting, then the values are adjusted so that their sum equals the `do` setting and all terms are displayed.

The adjustment is preferentially to `see1`. Suppose `do=7`, `see1=5`, `see2=4`. Then `see2` is left unchanged but `see1` is reduced to `7-4=3`. If, say, `do=7`, `see1=5`, `see2=8`, then `see2` is reduced to 7 and `see1` to `-1` (rather than zero, for technical reasons).

The default value for `see1` is 3; the default value for `see2` is 2.

Table 4.1: Settings for `\nmcRecur`

key	type	meaning	default
`do`	int≥ 0	number of terms to calculate	7
`see1`	int≥ 0	number of initial terms to display	3
`see2`	int≥ 0	number of final terms to display	2
`...`	chars	follow display of values with an ellipsis	

4.3.1.3 Ellipsis

Including three dots (periods, fullstops) in the settings option

```
...
```

ensures that a (proper) ellipsis is inserted after the final term is displayed. An example is provided by the display of the Fibonacci sequence at the start of this chapter. By default this option is turned off.

4.3.2 Form of result saved by `\nmcReuse`

In previous versions of `numerica-plus` it was possible to choose with the `reuse` setting what was saved by the next `\nmcReuse` command. The `reuse` setting has been discontinued in version 3 of `numeric-plus`. Now it is the last `see2` values of the recurrence sequence which are saved as a comma list in which each saved value is wrapped in braces in case the decimal comma is being used. (This setting has no effect when the star option is used with `\nmcRecur`. In that case only the numerical result of the final term calculated is saved.)

As an example,

```
\recur[env=multline*,p=.,vv@=1,do=11,see1=4,see2=2,
         vv={,}\\(vv)\\]
  { P_{n+2}(x)=\frac{1}{n+2}
        \Bigl(kxP_{n+1}(x)-(n+1)P_{n}(x)\Bigr) }
  [k=2n+3,n=1,P_{1}(x)=x,P_{0}(x)=1,x=0.5]
\reuse{legendre}
```

$\Longrightarrow$

$$P_{n+2}(x)=\frac{1}{n+2}\Bigl(kxP_{n+1}(x)-(n+1)P_n(x)\Bigr),$$
$$(k=2n+3, n=1, P_1(x)=x, P_0(x)=1, x=0.5)$$
$$\to 1,\ 0.5,\ -0.125,\ -0.4375,\ \ldots,\ -0.267899,\ -0.188229.$$

Now check to see what has been saved:

`$\legendre$` $\Longrightarrow -0.267899, -0.188229.$

As you can see, the final two (because of `see2=2`) of the 11 Legendre polynomials calculated ($P_0(x)$ is the first) have been saved.

4.3.2.1 Accessing individual terms

You may wish to gain access to individual members of the sequence of saved values. `numerica-plus` provides the utility function `\clitem` for this purpose.[1] It acts on the macro containing the comma list and a number specifying which member of the sequence is to be recovered: `1` for the first (leftmost) item, `2` for the second item, and so on.

---

[1] In fact a wrapper around the `expl3` function `\clist_item:Nn`.

`$\clitem\legendre 2$` $\Longrightarrow -0.188229$

The `$` delimiters ensure the minus sign displays correctly. Multi-digit numbers need to be enclosed in braces. Negative integers give access to the end of the sequence, `-1` for the last (rightmost) item, `-2` for the second last and so on. In the present case,

`$\clitem\legendre{-1}=\clitem\legendre 2$` $\Longrightarrow -0.188229 = -0.188229$.

4.3.3 Orthogonal polynomials

I've used Legendre polynomials in examples above, but orthogonal polynomials generally lend themselves to the `\recur` treatment. Quoting from *HMF* 22.7, orthogonal polynomials f_n satisfy recurrence relations of the form

$$a_{1n}f_{n+1}(x) = (a_{2n} + a_{3n}x)f_n(x) - a_{4n}f_{n-1}(x),$$

or in the standard form required by `\recur`,

$$f_{n+1}(x) = \frac{a_{2n} + a_{3n}x}{a_{1n}}f_n(x) - \frac{a_{4n}}{a_{1n}}f_{n-1}(x).$$

HMF 22.7 provides a listing of the coefficients a_{in} for the polynomials of Jacobi, Chebyshev, Legendre, Laguerre, Hermite and others, and tables for these polynomials.

For example, Laguerre polynomials satisfy the recurrence

$$L_{n+1}(x) = \frac{2n+1-x}{n+1}L_n(x) - \frac{n}{n+1}L_{n-1}(x).$$

with initial values $L_0(x) = 1$ and $L_1(x) = 1 - x$. So let's calculate the first 13 Laguerre polynomials for, say, $x = 0.5$:

```
\recur[env=multline*,do=13,see1=4,see2=2,
         vv={,}\\(vv)\\]
  { L_{n+1}(x)=\frac{2n+1-x}{n+1}L_{n}(x)-
    \frac{n}{n+1}L_{n-1}(x) }
       [L_{1}(x)=1-x,L_{0}(x)=1,x=0.5]
```

$\Longrightarrow$

$$\begin{multline*} L_{n+1}(x) = \frac{2n+1-x}{n+1}L_n(x) - \frac{n}{n+1}L_{n-1}(x),\\ (L_1(x) = 1 - x, L_0(x) = 1, x = 0.5)\\ \to 1,\ 0.5,\ 0.125,\ -0.145833,\ \dots,\ -0.313907,\ -0.23165 \end{multline*}$$

and for $x = 5$:

```
\recur[env=multline*,p=.,do=13,see1=4,see2=2,
         vv={,}\\(vv)\\]
  { L_{n+1}(x)=\frac{2n+1-x}{n+1}L_{n}(x)-
    \frac{n}{n+1}L_{n-1}(x) }
      [L_{1}(x)=1-x,L_{0}(x)=1,x=5]
```

$\Longrightarrow$

$$\begin{multline*} L_{n+1}(x)=\frac{2n+1-x}{n+1}L_{n}(x)-\frac{n}{n+1}L_{n-1}(x), \\ (L_{1}(x)=1-x,L_{0}(x)=1,x=5) \\ \to 1,\ -4,\ 3.5,\ 2.666667,\ \ldots,\ 0.107544,\ -1.448604. \end{multline*}$$

The results (reassuringly) coincide with those provided in *HMF* Table 22.11.

4.3.4 Nesting

It is possible to use the `\recur` command within an `\eval`, `\iter`, or `\solve` command, and indeed in `\recur` itself, but with this caveat: if `\recur` is nested within another command, the initial terms of the recurrence – e.g., $f_1 = 1, f_0 = 1$, for the Fibonacci series, or $L_1(x) = 1 - x, L_0(x) = 1$ for the Laguerre polynomials – must be located in the vv-list of that *inner* `\recur` command. Other shared variables can often be shifted to the vv-list of the outer command, but not these initial terms.

In the following example I multiply together (rather futilely) the third and fourth members of the sequence of Laguerre polynomials for $x = 5$ (the answer expected is `$ \eval{3.5\times2.666667} $` $\Longrightarrow$ 9.333334). Note that although it is tempting to shift the shared vv-lists of the inner `\recur*` commands to the vv-list of the outer `\eval` command, in fact only the `x=5` entry has been transferred:

```
\eval[p=.]{$
  \recur[do=3]
    { L_{n+1}(x)=\frac{2n+1-x}{n+1}L_{n}(x)-
        \frac{n}{n+1}L_{n-1}(x)}
      [L_{1}(x)=1-x,L_{0}(x)=1]
  \times
  \recur[do=4]
    { L_{n+1}(x)=\frac{2n+1-x}{n+1}L_{n}(x)-
        \frac{n}{n+1}L_{n-1}(x)}
      [L_{1}(x)=1-x,L_{0}(x)=1]
$}[x=5]
```

$\Longrightarrow 3.5\times 2.666667 = 9.333334.$

The terms of a recurrence relation are multi-token variables but `numerica` requires single tokens for its calculations. The problem for `\recur` is that the terms in the recurrence relation in the main (mandatory) argument differ from the terms in the vv-list: for instance f_n in the main argument, f_0 in the vv-list. If left like that, when `numerica` does its conversion from multi-token to single token variables, f_n would not be found since it differs from f_0. Hence a crucial first step for `\recur` is to reconcile the different forms, which it does by converting the forms in the vv-list to the forms in the recurrence in the main argument. To be available for this form change, they must reside in the *inner* vv-list. In the outer vv-list they would be inaccessible to the inner command.

***

This suggests an alternative way of proceeding: write the inital values of the recurrence terms in the *same* form in which they occur in the recurrence relation, together with an initial value for the recurrence variable, e.g., $f_{n+1} = 1, f_n = 1, n = 0$. This is not how mathematicians write the initial values in recurrence relations, which is why I did not pursue it, but it neatly sidesteps what is otherwise an initial awkwardness.

Chapter 5

Reference summary

5.1 Commands defined in `numerica-plus`

1. `\nmcIterate`, `\iter`
2. `\nmcSolve`, `\solve`
3. `\nmcRecur`, `\recur`
4. `\clitem`

5.2 Settings for the three main commands

5.2.1 Settings for `\nmcIterate`

Settings keys for `\nmcIterate`:

key	type	meaning	initial
`var`	token(s)	iteration variable	
`+`	int	fixed point extra rounding	`0`
`max`	int > 0	max. iteration count (fixed points)	`100`
`do`	int > 0	number of iterations to perform	`5`
`see`	int > 0	number of final iterations to view	`4`

5.2.2 Settings for `\nmcSolve`

Settings keys for `\nmcSolve`:

key	type	meaning	initial
`var`	token(s)	equation variable	
`dvar`	real $\neq 0$	initial step size	`1`
`+`	int	extra rounding	`0`
`max`	int > 0	max. number of steps before cut off	`100`

5.2.3 Settings for `\nmcRecur`

Settings keys for `\nmcRecur`:

key	type	meaning	initial
`do`	int≥ 0	number of terms to calculate	`7`
`see1`	int≥ 0	number of initial terms to display	`3`
`see2`	int≥ 0	number of final terms to display	`2`
`...`	chars	follow display of values with an ellipsis	