Aspects of Mathematical Modeling*

Attila Máté
Brooklyn College of the City University of New York

January 26, 2018

Contents

*Written for the course Mathematics 3202 (Mathematical Modeling) at Brooklyn College of CUNY.

1 Introduction to Maxima scripting

Maxima (you can click on the link to find the Wikipedia article on Maxima), is a computer algebra system (CAS) with powerful symbolic manipulation capabilities; these are similar to the capabilities of Mathematica or Maple. Brooklyn College has a site license of these latter two (that is, you can use them for free on a Brooklyn College computer), while Maxima is free software, licensed under GNU General Public License.[1.1] You can install Maxima on your computer by going to the Maxima Official Website.If you go to this site and click on Documentation at the top of the page, you will also find a Maxima Manual, the description of *Xmaxima*, a version of Maxima with GUI (graphical user interface), and other writings about Maxima. There are other sites as well, for exampla the site of Edwin L. (Ted) Woollett, especially the file mbelintro.pdf. In these notes, we will explain how to use Maxima in Linux, but Maxima is available also for Windows and the Macintosh operating system.[1.2]. The site lists the software installed on computers at the Library and the Library Cafe at Brooklyn College. Maxima is currently not listed; I requested that it be installed on some computers in early December, 2017, so it will perhaps soon be installed, or else it has already been installed but the website just mentioned listing software has not yet been installed. I have followed up on this issue, and I expect that it will be resolved soon. In any case, please feel free to ask for the software; if they realize that there is a demand for the software, they will hurry more with the installation.

1.1 The logistic difference equation

We illustrate Maxima scripting by the logistic difference equation:

$$x_{n+1} = r(1 - x_n)x_n, \tag{1.1}$$

where r is a parameter. We will discuss the logistic equation in more detail later. Given the value of r and value of x_0 (called an *initial value*), the value of x_n is determined for all $n \geq 0$. The dependence of x_n on the initial value and on the value of r is somewhat complicated, and numerical experiments are helpful to understand this behavior. Maxima can be used to carry out such exploration, but the purpose of the following scripts is mainly to explain Maxima programming.

[1.1] See the links to the explanations in the Introduction of the Wikipedia article, linked above.

[1.2] If you are running Maxima in Linux, usually the simpler option is to install Maxima from the repository of your Linux distribution

The first script we discuss calculates the first ten values of x_n determined by equation (1.1) and graphs the result:

```
1  kill(all);
2  linel : 60;
3  x(n) := if n=0 then initval else r*(1-x(n-1))*x(n-1)$
4  r : .6;
5  initval : 0.7;
6  mylist : makelist([n,float(x(n))],n,0,10);
7  plot2d([discrete,mylist],
8     [style, points],
9     [gnuplot_term,ps],
10    [gnuplot_ps_term_command,
11     "set term postscript eps \
12      solid lw 2 size 4 in, 3 in font \",18\""],
13    [gnuplot_out_file, "logistic.eps"]);
14 /*
15 I need to rewrite this without recursion, since it cannot
16 handle 20 points with recursion.
17 */
```

We called this file `logistic.max`; any Unix/Linux text editor can be used to create the file; our preference is vi, a text editor especially friendly to touch-typing. Note that a text editor is different from a word processor; for example, Microsoft Word is not a text editor. The number at the beginning of each line is not part of the file; it is the line number, making the discussion of the above script easier We run this script by the following command

```
$ maxima -b logistic.max > logistic.out
```

To execute this command, one simply needs to type it on the command line; note that the dollar symbol `$` at the beginning of the line is not to be typed; the sign is printed on the line by the computer to invite a command; this sign is called the *prompt* of the Linux command-line interface.

In actual life, the Linux prompt is usually not the symbol `$`, but this is the symbol most used in discussing the user prompt in the Unix/Linux literature. Linux is endlessly customizable, and the prompt is usually defined in the file `.bashrc` in the user's home directory. The purpose of the period at the beginning of the name of the file is to make the file invisible in normal directory listing. Here bash, is the Bourne Again Shell. A shell is a program that interprets the command typed by the user. The name Bourne Again Shell is a play of word on the Bourne Shell, an early Unix shell created by Stephen Bourne.

In the above command, `maxima` is the name of the command, `-b` specifies an option, that is, it gives a closer description as to how the command should behave, in the present case interpret the name following it as a *batch* file (i.e., a script). The symbol "`> logistic.out`" redirects the output of the command to the file `logistic.out`.[1.3] The listing of the file logistic.out is given next:

```
1
2  Maxima 5.27.0 http://maxima.sourceforge.net
```

[1.3]Without writing this, that is without redirection, the output would be directed to the *standard output*, i.e., it would be printed on the screen. Unix/Linux has two different types of output: standard output, i.e., what is normally printed on the screen, and the *standard error output*, listing errors, also printed on the screen, and normally indistiguishable from the standard output, except perhaps in its content. However, the standard output and the standard error output can be redirected to different destination. Whatever one types on the keyboard is normally the *standard input* (but the standard input is a more general concept).

```
 3  using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
 4  Distributed under the GNU Public License. See the file
 5  COPYING.
 6  Dedicated to the memory of William Schelter.
 7  The function bug_report() provides bug reporting
 8  information.
 9  (%i1)                          batch(logistic.max)
10
11  read and interpret file:
12  #p/home/mate/courses/modeling/maxima.ms/logistic.max
13  (%i2)                               kill(all)
14  (%o0)                                 done
15  (%i1)                              linel : 60
16  (%o1)                           60
17  (%i2) x(n) := if n = 0 then initval
18                                  else r (1 - x(n - 1)) x(n -
19  1)
20  (%i3)                      r : 0.6
21  (%o3)                        0.6
22  (%i4)                   initval : 0.7
23  (%o4)                        0.7
24  (%i5)  mylist : makelist([n, float(x(n))], n, 0, 10)
25  (%o5) [[0, 0.7], [1, 0.126], [2, 0.0660744],
26  [3, 0.037025144198784], [4, 0.021392569737506],
27  [5, 0.012560956618519], [6, 0.0074419073924081],
28  [7, 0.0044319152440625], [8, 0.0026473640227992],
29  [9, 0.001584213291918], [10, 9.4902213609822047E-4]]
30  (%i6) plot2d([discrete, mylist], [style, points],
31  [gnuplot_term, ps], [gnuplot_ps_term_command, set term post\
32  script eps      solid lw 2 size 4 in, 3 in font ",18"],
33  [gnuplot_out_file, logistic.eps])
34  (%o6)
35  (%o7)                    logistic.max
```

Again, the number at the beginning of each line is not part of the filei; it is the line number, allowing us to discuss the file more easily. We are going to discuss the above script and its output line-by-line. Line 1 of the output is blank, lines 2–8 describe Maxima itself, and line 9 echoes (i.e., repeats) the command "`batch(logistic.max)`".[1.4] What this refers to the fact that insted of running the the script logistic.max from the Linux command line as described above, we could have first started up maxima from the Linux command line as

```
$ maxima
```

and then entered the command `batch(logistic.max)`, or preferably, the command

```
batch("logistic.max")
```

after the Maxima input prompt. As seen from the output file above, the Maxima input lines start with the symbols `(%i1)`, `(%i2)`, `(%i3)`, ..., and the output lines are start with the symbols `(%o1)`,

[1.4] We used *logical* as opposed to *grammatical* placement of quotes, because in the kind of text we are writing it is important what exactly is being quoted. American (but not British) style rules require that a comma or a period should be placed before the closing quotation mark even if it is not part of the quoted text.

`(%o2)`, `(%o3)`, ..., (as one can see, a single maxima line may take up more than one line on the computer terminal) – one can call the symbol at the start of the input lines an *input prompt*; when running Maxima interactively, i.e., with the command

```
$ maxima
```

without any options, the input and outlines start similarly with the symbols described, but then nothing is printed on the input lines, and the user is invited to type a Maxima command. To quit a running a manually started instance of maxima, one needs to type

```
$ quit();
```

on the last input line. When running a file as a Maxima script, the input lines it echo (i.e., repeat) the input lines given in the input file (with certain changes, as we will describe below).

Line 1 if the script clears out all the earlier commands and values entered into Maxima. This plays no role if the script is run from the Linux command line, but it is important if it is run by the `batch` command in a running instance of Maxima. Line 2 sets the the variable `linel` to 60. Here the colon `:` is the assignment operator of Maxima; the variable on the left is given the value of the expression on the right. The variable `linel` is an interval Maxima variable specifying the lenght of (number of characters in) the output lines; the default value is 79. The only importance of setting it to 60 is to make the output lines short enough for the present manuscript, where 79 characters would not fit in one line of these writings. As for practical work, 79 characters per line are just fine. The commad is echoed on line 15 of the output, and 17 repeats the current value of the variable `linel`. The output on line 16 can be supressed by ending line 2 of the script with a dollar symbol `$a` instead of a semicolon `;`. Each Maxima command must be terminated with either a semicolon (inviting output) or a dollar sign (suppressing ouput); line breaks (instead of spaces) in Maxima commands make no difference. Going back to lines 11 and 12 of the output, these lines are self explanatory, except that line 12 gives the full path of the interpreted file in the Linux filesystem:

```
/home/mate/courses/modeling/maxima.ms/logistic.max
```

Note that the initial characters `#/p` in the line are not parts of this path; they added by Maxima to indicate that a path follows. Line 3 of the script gives the recursive formula given in equation (1.1). On the left-hand side we have `x(n)`, a function with function name `x` and argument `n`. The symbol `:=` is the function assignment symbol, and on the right-hand side of the symbol, the expression describes the vlaue to be assigned; the variables `initval` (initial value) and r will be given values later; the asterisk `*` is the multiplication symbol; the line ends with the dollar sign in order to suppress output. It is important to note that formula (1.1) had to be modified to express x_n rather than x_{n+1}, since we cannot assign to `x(n+1)` with the function assignent symbol `:=`. Observe that when this command is echoed on lines 17–19 of the output, the asterisk is replaced by a space. On lines 4 and 5 of the script we specify the values of the parameters `r` and `initval`. On line 6 we build a list of pairs (n, x_n). A pair is a list of length to, and the members of the list are separated by commasi; the list is enclosed in in square brackets. The command `float` indicates that one needs to take the numerical value (a floating point number) The command is terminated by a semicolon, so on lines 25–29 of the output you can see the list so generated. The second member of the last entry is the number `9.4902E-4` (we are omitting some digits), which indicates $9.4902 \cdot 10^{-4}$ in usual floating point notation. While Maxima is case sensitive, that is, it distinguishes lower and upper case letters, this floating number can equally well be written as `9.4902e-4`. Lines 7–13 give the command to dispay these pairs of numbers in the coordinate system; this is shown in Figure 1.1. The command

`plot2d` is the printing command; most of which are self-explanatory. The word `discrete` in the first argument indicates that what is to be displayed is a set of discrete[1.5] points. The program gnuplot is used in assisting the printing; On line 9, the file type displaying the plot is specified as PostScript: other file formats such as PDF or PNG can also be specified; PostScript is preferable in the preparation of the present manuscript; otherwise, one may prefer other formats for viewing or exchange. Lines 10–12 sets the window size for the plot at 4×3 inches, which is quite small, but fits well into the manuscript page. the important point here is that gnuplot does not allow the breaking up of the quoted text on lines 11 and 12 (even though Maxima itself allows line breaks instead of spaces). This is the reason that the end of line 11 is "quoted"; the backslash \ is a Linux quote character; here it precedes the new line character in the file (invisible when displaying the file), neutralizing the effect of the line break.[1.6] In normal circumstances it may not be necessary to break up this line, except help the appearance of this manuscript. Line 13 specifies the name of the plot file as `logistic.eps`, where `eps` stands for embedded (or encapsulated) PostScript.

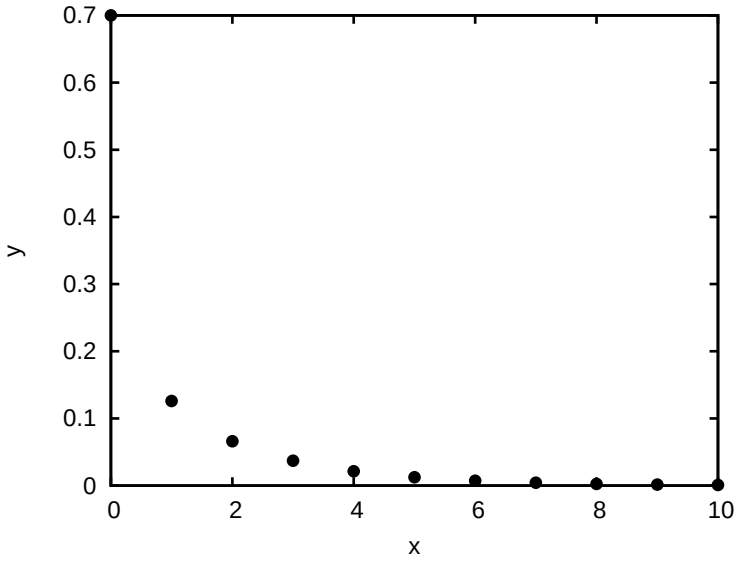

Figure 1.1: Logistic sequence

In the above program, x_n is evaluated recursively. It is not clear, however, when, for example x_4 is evaluated, the previously evaluated x_3 is used in the calculation, or whether full recursion is performed to get the value of x_4. or whether the full recursive calculation is started again with x_0. A further issue that might cause problems is that Maxima is used to perform floating point operations, so when `float(x(4))` is requested, it is likely that first the symbolic expression describing `x(4)` is calculated, and then the floating-point value of the expression is evaluated. Perhaps an optimizing compiler can go around some of these difficulties, but such an optimizing compiler is difficult to write, and it can cause unintended problems, so it is unlikely that the Maxima compiler would do

[1.5] It is important to note the difference between *discrete* and *discreet*. If you are uncertain, you can use several online dictionaries, such as Wiktionary.

[1.6] In Unix/Linux, lines are broken up by the newline character, denoted as `\n`, with hexadecimal ASCII code `0A`. Windows works differently, since the backslash is not a quote character in Windows, and Windows ends lines with different characters. See the site for details.

such an optimization.[1.7] In any case, the program can be written more efficiently with loops, given as the file `logistic_loop.max`. In the program we calculate x_n for $n = 0, 1, 2, \ldots, 20$. The loop version is much faster than the version using recursion; the recursive version is barely capable of calculating the first 20 or 30 values of x_n; the loop version does not flinch even when we calculate the first 100 members of the list. Here is the script:

```
kill(all);
linel : 60;
r : 1.6;
x : 0.7;
mylist : [];
loopto : 19;
for n : 0 step 1 thru loopto do (x_new : float(r*(1-x)*x),
  mylist : append(mylist,[[n,x]]), x : x_new);
n;
mylist : append(mylist,[[loopto+1,x_new]]);
plot2d([discrete,mylist],
   [style, points],
   [gnuplot_term,ps],
   [gnuplot_ps_term_command,
    "set term postscript eps \
     size 5 in, 3 in font \",24\""],
   [gnuplot_out_file, "logistic_loop.eps"]);
```

The output of this program is as follows. The program also creates a plot given in Figure 1.2

```

Maxima 5.27.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file
COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting
information.
(%i1)                     batch(logistic_loop.max)

read and interpret file:
#p/home/mate/courses/modeling/maxima.ms/logistic_loop.max
(%i2)                              kill(all)
(%o0)                                done
(%i1)                             linel : 60
(%o1)                               60
(%i2)                         r : 1.6
(%o2)                           1.6
(%i3)                         x : 0.7
(%o3)                           0.7
(%i4)                       mylist : []
(%o4)                            []
(%i5)                       loopto : 19
(%o5)                            19
(%i6) for n from 0 thru loopto
```

[1.7] The Kahan summation algorithm is a an interesting example of an algorithm that can be ruined by compiler optimization.

```
26 do (x_new : float(r (1 - x) x),
27 mylist : append(mylist, [[n, x]]), x : x_new)
28 (%o6)                          done
29 (%i7)                           n
30 (%o7)                           n
31 (%i8)  mylist : append(mylist, [[1 + loopto, x_new]])
32 (%o8) [[0, 0.7], [1, 0.336], [2, 0.3569664],
33 [3, 0.36726622283366], [4, 0.37181079103865],
34 [5, 0.37370804272938], [6, 0.37448054644601],
35 [7, 0.37479178684721], [8, 0.37491664537454],
36 [9, 0.37496664703303], [10, 0.37498665703334],
37 [11, 0.37499466252848], [12, 0.37499786496581],
38 [13, 0.37499914597903], [14, 0.37499965839045],
39 [15, 0.37499986335599], [16, 0.37499994534237],
40 [17, 0.37499997813694], [18, 0.37499999125478],
41 [19, 0.37499999650191], [20, 0.37499999860076]]
42 (%i9) plot2d([discrete, mylist], [style, points],
43 [gnuplot_term, ps], [gnuplot_ps_term_command,
44 set term postscript eps      size 5 in, 3 in font ",24"],
45 [gnuplot_out_file, logistic_loop.eps])
46 (%o9)
47 (%o10)               logistic_loop.max
```

We will discuss the new features of the above program. In the present organization of the program, the value of `r` and the initial value of the sequence are specified on lines 4 and 5 of the script, because they need be given before the loop on lines 7 and 8 of the program (we kept the initial value of x, but we chose a different value for r). The elements of the sequence are given as `x` and `x_new`. On line 5 the initial value of the list `mylist` of the pairs of coordinates of the point to be plotted is given as the empty list `[]`. The exit value of the loop is given as the variable `loopto` on line 6; we need this value for later use. In the loop on lines 7 and 8, the body of the loop needs to be enclosed in parentheses since several commands constitue the body of the loop, and the individual commands are separated by commas. The command `append` in the body of the loop on line 8 concatenates (merges) the lists given in the arguments (more than two arguments can be given. On line 9 of the script, we wrote `n` by itself to show that `n` has no numerical value at this point (in some other programming languages, one would expect that `n` had the exit value of the loop, probably 20). If `n` had a numerical value, its value would be printed on line 30 of the output. On line 10 of the script, we add the last pair to `mylist`. The line ends with a semicolon, and so the list is printed out on lines 32–41 of the output. If we wanted to suppress printing the list, we could have ended line 10 of the program with the dollar symbol `$` instead of a semicolon.

2 Fixed-point iteration

The equation $x = f(x)$ can often be solved by starting with a value $x = x_0$, and using the simple iteration $x_{n+1} = f(x_n)$ for $n \geq 0$. This method is called fixed-point iteration, and it is important for theoretical reasons. This is partly because other methods can often be reformulated in terms of fixed-point iteration. for example, when using Newton's method

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

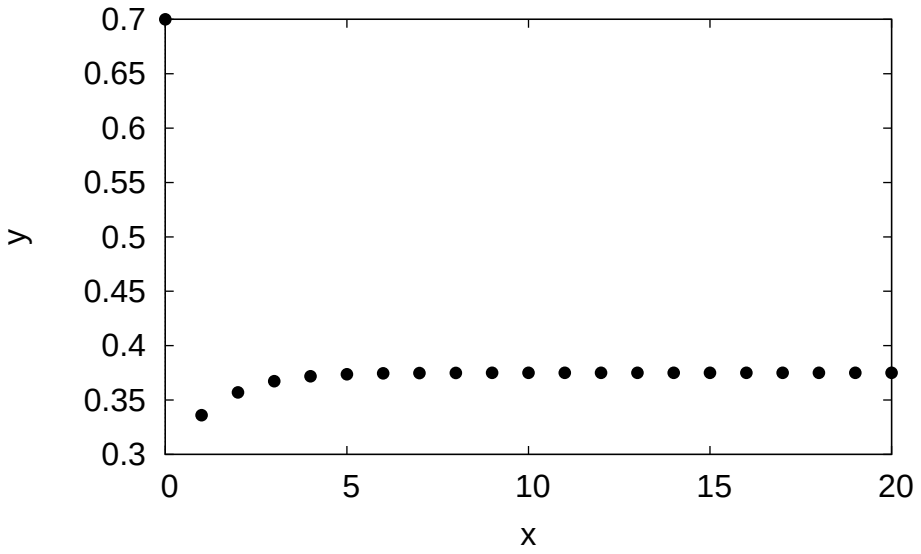

Figure 1.2: Logistic sequence, using loops

to solve the equation $g(x) = 0$, this can be considered as using fixed-point iteration to solve the equation

$$x = x - \frac{g(x)}{g'(x)}.$$

Furthermore, variants of fixed-point iteration are useful for solving certain systems of linear equations arising in practice (e.g., on account of cubic splines). A simple result describing the solvability of equations by fixed-point iteration is the following:

Theorem 2.1. *Assume $x = c$ is a solution of the equation $x = f(x)$. Assume further that there are numbers $r > 0$ and q with $0 \le q < 1$ such that*

$$|f'(x)| \le q \quad \textit{for all} \quad x \textit{ with } c - r < x < c + r.$$

Then, starting with any value $x_0 \in (c - r, c + r)$ and putting $x_{n+1} = f(x_n)$ for $n \ge 1$, the sequence $\{x_n\}_{n=1}^{\infty}$ converges to c.

When one tries to use this result in practice, the interval $(c-r, c+r)$ of course cannot be known exactly. However, the fact that $|f'(x)| < q$ in an interval for some q with $0 \le q < 1$ makes it reasonable to try to use fixed-point iteration to find a solution of the equation $x = f(x)$ in this interval.

Proof. Let $n \ge 0$, and assume that $x_n \in (c-r, c+r)$. By the Mean-Value Theorem of differentiation we have

$$f(x_n) - c = f(x_n) - f(c) = f'(\xi_n)(x_n - c),$$

where the first equality used the assumption that c is a root of $x = f(x)$, i.e., that $c = f(c)$; here ξ_n is some number between x_n and c. Clearly, we have $\xi_n \in (c - r, c + r)$, so we have $|f'(\xi_n)| \le q$. Therefore, noting that $x_{n+1} = f(x_n)$, we have

$$|x_{n+1} - c| \le q\,|x_n - c|. \tag{2.1}$$

Hence $x_{n+1} \in (c-r, c+r)$. Thus, given that $x_1 \in (c-r, c+r)$ by assumption, we can conclude that $x_n \in (c-r, c+r)$ for all positive integers n. Hence inequality (2.1) is valid for all positive integers n; thus we can conclude by induction that

$$|x_n - c| \leq q^n |x_0 - c|$$

for all integers $n \geq 0$. As q^n converges to 0 when n tends to zero, it follows that x_n converges to c. □

In addition to being used to solve equations numerically, fixed point iteration is also important in the study of mathematical models described by difference equations.

Definition 2.1. Given the equation

$$x_{n+1} = f(x_n) \qquad (n \geq 0), \tag{2.2}$$

the number c is called a *fixed point* of the equation if $f(c) = c$. The fixed point c is called *attractive* if there an open interval I containing c such that for all $t \in I$, for equation (2.2 with $x_0 = t$ we have $\lim_{n\to\infty} x_n = c$. The fixed point c is called *repulsive* if there is an open interval I containing t, for all $t \in I$ with $t \neq c$, for equation (2.2 with $x_0 = t$, the limit $\lim_{n\to\infty} x_n$ either does not exists, or if it exists and $\lim_{n\to\infty} x_n = c$ then $x_n = c$ for some $n > 0$.

The above theorem has the following simple corollary:

Corollary 2.1. *Given a real-valued function f and a real number c such that $f(c) = c$, and assume that $|f'(c)| < 1$ and that f' is continuous at c. Then c is an attractive fixed point of f.*

Proof. Let $q = \big(1 + |f'(c)|\big)/2$; then $q < 1$. The assumption that f' is continuous at c requires that c has a neighborhood in which f' exists. It also follows from the same assumption that there is a neighborhood of c in which $|f'(x)| < 1$. Hence the result follows from Therem fixed: conv theorem. □

A partial converse to this is the following

Theorem 2.2. *Assume $x = c$ is a solution of the equation $x = f(x)$. Assume further that there are numbers a, b with $a < c < b$ such that*

$$|f'(x)| \geq 1 \quad \textit{for all} \quad x \textit{ with } a < x < b.$$

Then starting with any value x_0 and putting $x_{n+1} = f(x_n)$ for $n \geq 0$, the sequence $\{x_n\}_{n=0}^{\infty}$ does not converge to c unless $x_k = c$ for some positive integer k.

Of course, if $x_k = c$ for some positive integer k, then $x_n = c$ for all $n \geq k$. However, it is very unlikely that we accidentally end up with $x_k = c$ in practice, and so, if $|f'(x)| > 1$ near the solution of the equation $x = f(x)$, using fixed-point iteration to find the solution should be considered hopeless.

Proof. Assuming x_n converges to c, we must have a positive integer N such that $x_n \in (a, b)$ for every $n \geq N$. So, for any $n \geq N$ we have, by the Mean-Value Theorem that

$$f(x_n) - c = f(x_n) - f(c) = f'(\xi_n)(x_n - c),$$

for some $\xi_n \in (a, b)$, and so, noting that $x_{n+1} = f(x_n)$ and that $|f'(\xi_n)| \geq 1$, we obtain that

$$|x_{n+1} - c| \geq |x_n - c|$$

for all $n \geq N$. Thus we have

$$|x_n - c| \geq |x_N - c|$$

for all $n \geq N$. Hence x_n cannot converge to c unless $x_N = c$. □

Corollary 2.2. *Given a real-valued function f and a real number c such that $f(c) = c$, and assume that $|f'(c)| > 1$ and that f' is continuous at c. Then c is a repulsive fixed point of f.*

Proof. The assumptions imply that c has a neighborhood in which $|f'(c)| > 1$. Hence the conclusion follows from Theorem 2.2. □

Fixed point iteration can be visually illustrated by the cobweb diagram. For more pictures, see here or the result of a Google search with key words "fixed point cobweb diagram" (without the quotes) here.

3 Some results on the stability of difference equations

We will further study the fixed point difference equation (2.2). In particular, we will assume that f is a real-valued continuous function statisfying the following conditions stated in [7, §5.F, p. 365]:

Definition 3.1 (Meyer conditions)**.** Let f be a continuous functionfunction $\mathbb{R} \to \mathbb{R}$. We say that f satisfies the *Meyer conditions* if the following hold

(a) We have $f(0) = 0$ and there is a $w > 0$ such that $f(x) > 0$ for x with $0 < x < w$ and $f(w) = 0$.

(b) f has maximum M in the interval $[0, w]$ at a single point $x_{\max}$. For $x \in \mathbb{R}$ with $x < x_{\max}$, f is increasing, and for $x \in \mathbb{R}$ with $x > x_{\max}$, f is decreasing.

(c) For $x > 0$, there is exactly one value of x for which $f(x) = x$; call this value R. We have $R < w$.

(d) For x with $0 < x < R$ we have $f(x) > x$.

The discussion of the consequences of the Meyer conditions follows [7, §5.F, p. 365ff.]:[3.1]

Theorem 3.1. *Let f be a continuous function satisfying the Meyer conditions in Definition 3.1, and assume $R \leq x_{\max}$ Then, with $x_0 \in (0, w)$ and x_n defined by equation (2.2), we have $\lim_{n\to\infty} x_n = R$.*

Proof. According to Meyer conditions (a), and (b), f maps the interval $(0, w)$ into $(0, M]$, since M is the maximum of f on the interval $(0, w)$. As $R \leq x_{\max}$, we have $M = f(x_{\max}) \leq x_{\max}$. Indeed, we have $f(x) < x$ for all $x > R$; this is because $f(w) = 0$, f is continuous, and $f(x) = x$ holds for $x \in (0, w)$ only if $x = R$. Thus $(0, M] \subset (0, x_{\max}]$, and so $x_n \in (0, x_{\max}]$ for all $n \geq 1$. Since $f(x)$ is increasing on the interval $[0, x_{\max}]$, if $x_n \leq x_{n+1}$ for some $n \geq 1$, we also have $x_{n+1} = f(x_n) \leq f(x_{n+1}) = x_{n+2}$, and so the, by induction, sequence $\{x_k\}_{k=n}^{\infty}$ is nondecreasing. Similarly, if $x_n \geq x_{n+1}$ for some $n \geq 1$, we also have $x_{n+1} = f(x_n) \geq f(x_{n+1}) = x_{n+2}$, and so the, by induction, sequence $\{x_k\}_{k=n}^{\infty}$ is nonincreasing. Since this sequence is bounded, it has a limit; for the limit c, we must have $f(c) = c$. We cannot have $c = 0$, since if $x_n \in (0, R)$, then $x_{n+1} = f(x_n) > x_n$, so the limit cannot be 0. The only other fixed point is $c = R$. Hence $\lim_{n\to\infty} x_n = R$. □

[3.1] The abbreviation "ff", usually followed by a perion, means "and the following pages."

3.1 Stability of the logistic equation

The logistic difference equation was already introduced in Subsection 1.1. It is the

$$x_{n+1} = r(1 - x_n)x_n \qquad \text{for all} \quad n \geq 0, \tag{3.1}$$

where r is some parameter. We have

Corollary 3.1. *Assume $1 < r < 2$, $0 < x_0 < 1$, and x_n for $n \geq 0$ satisfies equation (3.1). Then*

$$\lim_{n\to\infty} x_n = \frac{r-1}{r}.$$

Proof. Writing $f(x) = r(1-x)x$, we $f'(x) = r(1-2x)$. It is easy to check that the Meyer conditions of Definition 3.1 are satisfied with $w = 1$, $x_{\max} = 1/2$, and $R = (r-1)/r$. Indeed,

$$0 < R = \frac{r-1}{r} < \frac{1}{2} = x_{\max},$$

since $1 < r < 2$. Hence the result follows from Theorem 3.1. □

4 Fixed points of linear matrix difference equations

4.1 Reducing the inhomogeneous equation to a homogeneous equation

Consider the equation

$$\mathbf{x}_{n+1} = A\mathbf{x}_n + \mathbf{b}, \qquad (n \geq 0) \tag{4.1}$$

where A is a $\nu \times \nu$ matrix, and $\mathbf{x}_n$ and $\mathbf{b}$ are vectors of size ν (i.e., $1 \times \nu$ matrices). The question we are going to consider is whether there is a vector $\mathbf{v}$ such that

$$\lim_{n\to\infty} \mathbf{x}_n = \mathbf{v}.$$

Here the limit may be defined componentwise, or with the aid of a vector norm. For the considerations in this sections, we will allow A and $\mathbf{x}_n$ to have complex entries.

If such a $\mathbf{v}$ exits, we must have $\mathbf{v} = A\mathbf{v} + \mathbf{b}$. Hence equation (4.1) can be written as

$$\mathbf{x}_{n+1} - \mathbf{v} = A(\mathbf{x}_n - \mathbf{v}).$$

Thus, the case of the inhomomogeneous equation (4.1) can be reduced to the case of the homogeneous equation with the substitution $\mathbf{y}_n = \mathbf{x}_n - \mathbf{v}$.

4.2 Fixed points of the homogeneous equation

The homogeneous equation is given by the case $\mathbf{b} = 0$ of equation (4.1):

$$\mathbf{x}_{n+1} = A\mathbf{x}_n, \qquad (n \geq 0) \tag{4.2}$$

A vector $\mathbf{v}$ is called a fixed point of this equation if the equation is satisfied with $\mathbf{x}_n = \mathbf{v}$ for all $n \geq 0$. Such a vector will be called a *stable fixed point* if for all initial valued $\mathbf{x}_n$, the solution of equation (4.2) converges to $\mathbf{v}$.[4.1]

[4.1] One might want to require this only for vectors $\mathbf{x}_0$ that are in some sense close to $\mathbf{v}$; however, as we will see such a definition does not make good sense.

The vector 0 is always a fixed point of equation (4.2). For a vector $\mathbf{v} \neq 0$ to be a fixed point is equivalent to saying that $A\mathbf{v} = \mathbf{v}$, i.e., that $\mathbf{v}$ is an eigenvector of A associated with the eigenvalue 1. If this is the case, the vector $\mathbf{v}$ is not is not a stable fixed point, since for any complex number α we have $A(\alpha\mathbf{v}) = \alpha\mathbf{v}$. To discuss the condition for the vector 0 to be a stable fixed point we need the following

Definition 4.1. The spectral radius of the square matrix A is the maximum of the absolute values of its eigenvalues.

The condition for 0 to be a stable fixed point of equation equation (4.2) is given by the following

Theorem 4.1. *The vector* 0 *is a stable fixed point of equation (4.2) if and only if the spectral radius of A is less than* 1*.*

This is a consequence of the following

Theorem 4.2 (Gelfand's formula)**.** *If $\|\cdot\|$ is a matrix norm compatible with a vector norm,* [4.2] *for any square matrix A, radius, we have*

$$\rho(A) = \lim_{n\to\infty} \|A^n\|^{1/n},$$

where $\rho(A)$ denotes the spectral radius of A.

The proof of this can be carried out using the *Jordan normal form*, [4.3] but a more elegant proof, generalizable way beyond the scope of matrices, uses complex analysis. This proof, for matrices, is given in [2, Section 43, pp. 222–223];

Theorem 4.1 is an immediate consequence of Theorem 4.2, since $\rho(A) < 1$ implies that $\|A^n\| \to 0$ and so $\|A^n\mathbf{x}\| \to 0$ for all $\mathbf{x}$ as $n \to \infty$. On the other hand, if $\rho(A) \geq 1$ then there is an eigenvector $\mathbf{x}$ of A with an eigenvalue λ with $|\lambda| \geq 1$, and so for which

$$\|A^n\mathbf{x}\| = \|\lambda^n\mathbf{x}\| = |\lambda|^n\|\mathbf{x}\| \geq \|\mathbf{x}\| > 0$$

for all $n \geq 0$.

Corollary 4.1. *Equation (4.1) has a stable fixed point if and only if the spectral radius of A is less than* 1*.*

This corollary applies both to the inhomogeneous and to the homogeneous equation; the latter is represented by the case $\mathbf{b} = 0$.

4.3 The case of multiple fixed points

Assume $\mathbf{u}$ and $\mathbf{v}$ are fixed points of equation (4.1), that is, $A\mathbf{u} = \mathbf{u} + \mathbf{b}$ and $A\mathbf{v} = \mathbf{v} + \mathbf{b}$, where $\mathbf{u} \neq \mathbf{v}$. Then $A\mathbf{w} = \mathbf{w}$ with $\mathbf{w} = \mathbf{u} - \mathbf{v} \neq 0$; that is, 1 is an eigenvalue of A. Then the spectral radius of A is ≥ 1, and so 0 is not a stable fixed point of equation (4.2), and so equation (4.1) has no stable fixed point.[4.4]

[4.2] A matrix norm $\|\cdot\|$ is called compatible, or *consistent*, with a vector norm, also denoted as $\|\cdot\|$, if for any matrix A and any column vector $\mathbf{x}$ we have

$$\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|.$$

In the discussion that follows, $\|\cdot\|$ will denote a fixed vector norm or a fixed matrix norm compatible with this vector norm.

[4.3] For the Jordan normal form of matrices, see [3, Subsection 8.7, on p. 20].

[4.4] The assertion that 1 is an eigenvalue of A means the same thing as A is singular. If A is singular, then the equatinon $\mathbf{x} = A\mathbf{x} + \mathbf{b}$ either has no solutions or it has infinite many solutions, depending on the choice of $\mathbf{b}$.

4.4 An example

The book [1, Section 1.4, Example 4, p. 47] gives the system of equations

$$\begin{aligned} S_{n+1} &= 0.75S_n + 0.20U_n + 0.40A_n, \\ U_{n+1} &= 0.05S_n + 0.60U_n + 0.20A_n, \\ A_{n+1} &= 0.20S_n + 0.20U_n + 0.40A_n, \end{aligned} \tag{4.3}$$

where $n \geq 0$. The fixed points of this equation are analyzed for stability numerically under the condition that that $S_0 + U_0 + A_0 = 4000$. It then follows that

$$S_n + U_n + A_n = 4000 \qquad \text{for all} \quad n \geq 0; \tag{4.4}$$

indeed, if we add the equations in (4.3), we obtain

$$S_{n+1} + U_{n+1} + A_{n+1} = S_n + U_n + A_n \qquad \text{for all} \quad n \geq 0. \tag{4.5}$$

We will incorporate equation (4.4) into the system (4.3) by subtracting an appropriate multiple of the equation

$$0 = S_n + U_n + A_n - 4000 \tag{4.6}$$

from each equation of the system so as to eliminate A_n from the right-hand side (for the first equation, we need to multiply the this equation by 0.40, for the second, by 0.20, for the third, by 0.40). We obtain the system

$$\mathbf{x}_{n+1} = A_1\mathbf{x}_n + \mathbf{b}_1, \tag{4.7}$$

where [4.5]

$$A_1 = \begin{pmatrix} 0.35 & -0.2 & 0.0 \\ -0.15 & 0.4 & 0.0 \\ -0.2 & -0.2 & 0.0 \end{pmatrix} \quad \text{and} \quad \mathbf{b}_1 = \begin{pmatrix} 1600 \\ 800 \\ 1600 \end{pmatrix},$$

and $\mathbf{x}_n = (S_n, U_n, A_n)^T$.[4.6] The fixed point of this equation is

$$\mathbf{x} = (-2222.22\ldots, -777.77\ldots, 1000)^T.$$

The eigenvalues of the matrix A_1 are .55, .2, and 0; so the spectral radius of A_1 is .55. According to Corollary 4.1, the fixed point of equation (4.7) is stable.

4.4.1 A note on hand calculations

As we will explain in the next section, we used Maxima to assist in the calculations, even though hand calculations would also have been quite easy. Indeed, the matrix A_1 and the column vector $\mathbf{b}_1$ in equation (4.7) can be calculated quite easily. by hand. We can get away with less, since we do not need the column vector $\mathbf{b}_1$ in order to see that the fixed point of the system (4.3) is stable; the

[4.5] The symbol A_n in equation (4.3) and name of the matrix A_1 used in equation (4.7) conflict; however, the context clarifies which is meant. It would be more troublesome for us to change the notation in equation (4.7) so as to avoid this conflict than whatever misunderstanding may result from not doing so.

[4.6] C^T denotes the transpose of the matrix C. To save place, one often writes a column vector as the transpose of a row vector.

only thing we need are the eigenvalues of the matrix A_1, and we will see, we only need the first two rows of the matrix A_1 for this.

For a square matrix M, its eigenvalues are the zeros of the polynomial $\det(M - \lambda I)$ of λ, called the characteristic polynomial of M; here I is the identity matrix of the same size as M, and $\det(C)$ stands for the determinant of the matrix C. That is, we need the zeros of the polynomial

$$\det(A_1 - \lambda I) = \begin{vmatrix} 0.35-\lambda & -0.2 & 0.0 \\ -0.15 & 0.4-\lambda & 0.0 \\ -0.2 & -0.2 & -\lambda \end{vmatrix} = -\lambda \begin{vmatrix} 0.35-\lambda & -0.2 \\ -0.15 & 0.4-\lambda \end{vmatrix},$$

where the second equation follows by expanding the determinant in the middle member of these equations with respect to its last column. The polynomial on the right-hand side is zero if $\lambda = 0$ or if the 2×2 determinant on the right-hand side is zero. This determinant is a quadratic polynomial of λ, and to find out when this quadratic polynomials is zero is not too hard. Indeed, this quadratic polynomial is

$$\lambda^2 - .75\lambda + .11 = \frac{1}{100}(100\lambda^2 - 75\lambda + 11).$$

The discriminant of the polynomial in parentheses is

$$75^2 - 4 \cdot 100 \cdot 11 = 1225 = 35^2,$$

showing that the zeros of this polynomial are quite easy to find by hand. They are

$$\frac{75+35}{200} = \frac{110}{200} = \frac{11}{20} \qquad \text{and} \qquad \frac{75-35}{200} = \frac{40}{200} = \frac{1}{5}$$

according to the quadratic formula.

4.5 The Maxima program supporting the above example

We will next discuss the Maxima program used to support the calculations described in Subsection 4.4. Here is the script `matrix_fix.max` that we used:

```
kill(all);
linel : 60;
load(eigen);
ratprint : false$
a : matrix(
    [0.75, 0.20, 0.40],
    [0.05, 0.60, 0.20],
    [0.20, 0.20, 0.40]
)$
determinant(a);
id : identfor(a)$
d : matrix([1,1,1]);
acol3 : transpose(transpose(a)[3]);
tosubtract : acol3 . d;
a1 : a - tosubtract;
b1 : 4000*acol3;
fixpt : -(a1-id)^^(-1).b1;
float(eigenvalues(a1));
```

The output of this script is as follows:

```
1
2  Maxima 5.27.0 http://maxima.sourceforge.net
3  using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
4  Distributed under the GNU Public License. See the file
5  COPYING.
6  Dedicated to the memory of William Schelter.
7  The function bug_report() provides bug reporting
8  information.
9  (%i1)                         batch(matrix_fix.max)
10
11 read and interpret file:
12 #p/home/mate/courses/modeling/maxima.ms/matrix_fix.max
13 (%i2)                                kill(all)
14 (%o0)                                  done
15 (%i1)                               linel : 60
16 (%o1)                              60
17 (%i2)                        load(eigen)
18 (%o2) /usr/share/maxima/5.27.0/share/matrix/eigen.mac
19 (%i3)                     ratprint : false
20                            [ 0.75  0.2  0.4 ]
21                            [                ]
22 (%i4)                 a : [ 0.05  0.6  0.2 ]
23                            [                ]
24                            [ 0.2   0.2  0.4 ]
25 (%i5)                     determinant(a)
26 (%o5)                          0.11
27 (%i6)                    id : identfor(a)
28 (%i7)                    d : [ 1  1  1 ]
29 (%o7)                      [ 1  1  1 ]
30 (%i8)           acol3 : transpose(transpose(a) )
31                                                3
32                               [ 0.4 ]
33                               [     ]
34 (%o8)                         [ 0.2 ]
35                               [     ]
36                               [ 0.4 ]
37 (%i9)                 tosubtract : acol3 . d
38                         [ 0.4  0.4  0.4 ]
39                         [               ]
40 (%o9)                   [ 0.2  0.2  0.2 ]
41                         [               ]
42                         [ 0.4  0.4  0.4 ]
43 (%i10)                 a1 : a - tosubtract
44                        [  0.35   - 0.2  0.0 ]
45                        [                    ]
46 (%o10)                 [ - 0.15   0.4   0.0 ]
47                        [                    ]
48                        [ - 0.2   - 0.2  0.0 ]
49 (%i11)                    b1 : 4000 acol3
50                              [ 1600.0 ]
51                              [        ]
52 (%o11)                       [ 800.0  ]
```

```
53                              [         ]
54                              [ 1600.0  ]
55                                        <- 1>
56  (%i12)         fixpt : (- (a1 - id)     ) . b1
57                       [ 2222.222222222223 ]
58                       [                   ]
59  (%o12)               [ 777.7777777777778 ]
60                       [                   ]
61                       [      1000.0       ]
62  (%i13)               float(eigenvalues(a1))
63  (%o13)       [[0.55, 0.2, 0.0], [1.0, 1.0, 1.0]]
64  (%o13)                    matrix_fix.max
```

We will give a line-by-line description of above the script. Occasianally, we may also comment on the output as well, though most of it is self-explanatory. Line 1 of the script specifies the maximum length of the output lines, and line 3 loads the Maxima package `eigen` needed in eivenvalue calculations. Line 4 makes the logical variable (i.e., a variable that can assume the values *true* or *false*) `ratprint` false, thereby suppressing messages informing about the conversion of floating point numbers to rational numbers (i.e., common fractions); there are many such messages that would make the output excessively long. At the end of the line, the dollar sign `$` suppresses an output message that the variable `ratprint` was indeed given the truth value false. Lines 5–9 specify the matrix `a`, the coefficient matrix in equation (4.3). The way the matrix was entered, with line breaks at the end of the rows may be visually pleasing, but it is irrelevant as far as Maxima is concerned. On line 10, the determinant of the matrix `a` is checked to see that the rows of the matrix are linearly independent – even though this is not of any importance in our discussions: they indeed are, since line 26 of the output shows that this determinant is 0.11, that is, it is different from 0. Line 11 creates the identity matrix `id` of the same size as the matrix `a`. On line 12, `d` is defined as the row vector $(1, 1, 1)$, the coefficient vector in equation (4.6).

On line `acol3` is defined and the third column of the matrix `a`; this vector contains the appropriate multiples of equation (4.6) that need to be subtracted form the equations of system (4.1). The way a matrix is entered (cf. lines 5–9 of the input), the arguments of a matrix are its rows, so there is an easy way to get a specific column of the matrix. For this reason, we take the transpose of the matrix `a`, take the third row of it, and then to get a column vector, we take the transpose of this row vector.[4.7] As indicated, the matrix `tosubtract` described on line 14 of the script is the matrix product of the column vector `acol3` and the row vector `d`; the dot `.` on the line is the symbol for matrix product. The resulting matrix is shown on lines 37-42 of the output. We obtain the matrix `a1`, corresponding to the coefficient matrix A_1 in equation (4.7) by subtracting this matrix from the matrix `a`, corresponding to the coefficient matrix in equation (4.2). The column vector $\mathbf{b}_1$ in equation (4.7) is defined on line 16 of the script; the asterisk `*` on the line is used to multiply a matrix by a scalar.[4.8] The fixed point of equation (4.7) is calculated on line 17; here the symbol `^^(-1)` indicates matrix inversion (that is, a matrix raised to the power -1; other integer powers are also allowed). On line 18, the eigenvalues of the matrix `a1` are calculated; line 63 of the output shows the result. The first triple indicates the eigenvalues, and the second triple gives the corresponding multiplicities. One would expect these multiplicities `[1.0, 1.0, 1.0]` to be integers, which they were before converting the output to floating point on line 18 of the script.

[4.7] If we wanted to get the entry in the second row and the first column of the matrix, we would need to enter `a[2][1]`.

[4.8] The use of the asterisk `*` for matrices is more general, but this is not of immediate interest to us – see the manual for details; it definitely not used to indicate matrix multiplication, for which the dot `.` is used, as mentioned.

5 Random numbers and Monte Carlo methods

Calculations using random numbers occur frequently in computations. The problems where they are used may involve simulation of truly random processes, or they may involve computational methods involving randomness even though the problem to be solved are deterministic in nature. The broad class of mathematical algorithms using random numbers are called Monte Carlo methods. Random numbers are also used in many algorithms involving computer security and cryptographic key exchange.

The numbers used in these algorithms are not truly random, they just appear random; such numbers are called pseudorandom numbers. The reason for not using truly random numbers are several. First, it is difficult to find truly random processes that are capable of generating such numbers is a really balanced way, that is, without bias, and with a speed required in computer calculations. Further, during program testing, it is often important to carry out the same calculation repeatedly, but this would be impossible if actual random numbers were used.

Random numbers used in cryptography have additional requirements often not needed in usual calculations, since such numbers need to withstand methods of discovery by a sophisticated adversary. Random numbers suitable for this purpose are called cryptographically secure pseudorandom numbers. Pseudorandom numbers used in mathematical modeling or in scientific calculations are unlikely to be cryptographically secure. The next program illustrates random numbers in Maxima.

```
kill(all);
linel : 60;
load(distrib);
st : make_random_state(56088371546)$
set_random_state(st);
random_normal(2,3,10);
set_random_state(st);
random_normal(2,3);
random_normal(2,3);
set_random_state(st);
random_normal(2,3);
random_normal(2,3);
```

The output of this program appears next.

```

Maxima 5.27.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file
COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting
information.
(%i1)                          batch(random.max)

read and interpret file:
#p/home/mate/courses/modeling/maxima.ms/random.max
(%i2)                               kill(all)
(%o0)                                 done
(%i1)                              linel : 60
(%o1)                         60
(%i2)                  load(distrib)
```

```
18 (%o2) /usr/share/maxima/5.27.0/share/distrib/distrib.mac
19 (%i3)       st : make_random_state(56088371546)
20 (%i4)              set_random_state(st)
21 (%o4)                      done
22 (%i5)            random_normal(2, 3, 10)
23 (%o5) [6.165594866462493, 0.13110245802358,
24 - 1.519759347422934, 3.718967001669206, 4.887487921863613,
25 1.638569730008628, 3.706855370154149, 2.785092006771531,
26 5.363605674459343, 1.145985458146]
27 (%i6)              set_random_state(st)
28 (%o6)                      done
29 (%i7)              random_normal(2, 3)
30 (%o7)                 1.145985458146
31 (%i8)              random_normal(2, 3)
32 (%o8)               5.363605674459343
33 (%i9)              set_random_state(st)
34 (%o9)                      done
35 (%i10)             random_normal(2, 3)
36 (%o10)                1.145985458146
37 (%i11)             random_normal(2, 3)
38 (%o11)              5.363605674459343
39 (%o11)                   random.max
```

We will comment on the above script and the resulting output line by line. On line 3 of the script, the maxima package `distrib` is loaded; this package makes it possible to generate random numbers following a large number of of distributions occurring in mathematics and statistics. On line 4 a state of the random number generator is created; this state is generated from a seed, the integer argument entered on this line of the function `make_random_state`. This state is saved in the variable `st`, and it is reused a number of times in the script. In fact, on lines 5, 7, and 10 of the script, the random number generator is assigned the state `st`. The effect of this is that exactly the same random numbers are generated in the output as we will explain.

The command on line 6 of the script generates a sequence of 10 random floating point numbers distributed according to a normal dstribution of mean 2 and standard deviation 3. on lines 8, 9, 11, and 12 each, a single random number following the same normal distribution is generated. It is to be noticed that the two random numbers on line 26 are exactly the same as the numbers on lines 30 and 32, and then again on lines 36 and 38 of the output. This is because we reset the state of the random number generator to the same state.

5.1 A Monte Carlo calculation of the area of the circle

The next script uses a Monte Carlo method to approximate the area of the part lying in the first quadrant of the unit circle with center at the origin. The method is randomly select points following uniform distribution in the unit square with vertices $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, and counting the proportion that falls in the part of the circle.

```
1 kill(all);
2 linel : 60;
3 st : make_random_state(56088371546)$
4 set_random_state(st);
5 count : 0;
6 tries : 10000;
7 for n : 1 step 1 thru tries do ( x : random(1e5)/1e5,
```

```
y : random(1e5)/1e5, count : if x^2+y^2<1 then count+1 else
count )$
mtc_area : float(count/tries);
true_area: float(%pi/4);
abserr : mtc_area-true_area;
relerr : abserr/true_area;
```

The output of this script as follows.

```

Maxima 5.27.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file
COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting
information.
(%i1)                         batch(area_circle.max)

read and interpret file:
#p/home/mate/courses/modeling/maxima.ms/area_circle.max
(%i2)                                 kill(all)
(%o0)                                   done
(%i1)                                linel : 60
(%o1)                            60
(%i2)       st : make_random_state(56088371546)
(%i3)                 set_random_state(st)
(%o3)                           done
(%i4)                      count : 0
(%o4)                           0
(%i5)                    tries : 10000
(%o5)                       10000
                                    random(100000.0)
(%i6) for n thru tries do (x : ----------------,
                                        100000.0
    random(100000.0)
y : ----------------, count :
        100000.0
    2    2
if y  + x  < 1 then 1 + count else count)
                                        count
(%i7)              mtc_area : float(-----)
                                        tries
(%o7)                          0.7943
                                         %pi
(%i8)               true_area : float(---)
                                          4
(%o8)                   0.78539816339745
(%i9)           abserr : mtc_area - true_area
(%o9)                 0.0089018366025517
                                   abserr
(%i10)                relerr : ---------
                                 true_area
```

```
45  (%o10)                  0.01133417038314
46  (%o10)                  area_circle.max
```

We give a line-by-line description of the script, occasionally commenting also on the output. On lines 3–4, the state of the random number generator is set. On line 6, the number of tries is set to be $100,000$. The loop on lines 7–9 counts the number of those falling into the unit circle. On line 8 in the loop, a random integer in the range 0 to `1e5`, the latter being the computer notation for the floating point number $1 \times 10^5 = 100,000$ is created, and this is immediately divided by `1e5`, to get a random number uniformly distributed in the interval $[0, 1]$. On line 10, the variable `mtcarea`, abbreviating Monte-Carlo area, is assigned the ration of successful tries divided by the total number of tries. On line 11, the true area is evaluated as $\pi/4$. Lines 12 and 13 calculate the absolute and relative errors are are calculated.

6 The middle square random number generator

6.1 The classic middle-quare method

Th classic middle square method was invented by John von Neumann in 1946 to generate preuso-random numbers by taking an 4-digit integer as a starting value. Squaring this number produces an 8-digit number (occasionally, there are fewer digits that should be supplemented to 8 by adding leading zeros); taking the middle 4 digits gives the first random number; squaring this and taking the 4 middle digits gives the second one. This can be repeated to get a series of random numbers. The method was fast but it had the problem of often producing short cycles, and it also had issues with the accumulation of zero digits in the random numbers, which will not disappear in subsequent iterations. Von Neumann well understood the weaknesses of his method, but he justified its use by its speed and by noting that it will be noticed immediately when the method went awry.[6.1] The method was important in early calculations concerning nuclear processes, such as simulating the absorption of neutrons by shielding materials. At the time, the only electronic computer available to do these calculations was the ENIAC, and the simplicity of the method was also an advantage.[6.2]

6.2 The updated middle square method: a C++ implementation

The problems with the middle square methods can be corrected, as shown in [11].[6.3], and the resulting method is one of the fastest method of producing random numbers to date. The method starts with a large integer, squares it, takes the middle digits of the square;[6.4] It then adds successive terms of a sequence known to be equidistributed modulo a large integer.[6.5] These steps are repeated to produce a succession of random numbers. Adding the successive terms of the equidistributed sequence eliminates the defects of von Neumann's original method.

[6.1] See the History section of the Wikipedia article on the Monte Carlo method.

[6.2] The linked Wikipedia article about the ENIAC is well worth reading for those interested in the history of computing.

[6.3] The title of the article is "Middle square Weyl sequence RNG." Here "RNG" referes to Random Number Generator, but the reference to "Weyl sequence" is a somewhat obscure reference to the famous mathematician Hermann Weyl. Where there is a *Weyl sequence* in the theory of Hilbert space operators, this is unlikely to be referred to. The reference is probably to Weyl's equidistribution article [10], though the connetion to this article is remote.

[6.4] In fact, it takes only the lower half of the middle digits. The reason for this is the way compters are constructed, and not some theoretical reason, as we will see.

[6.5] This is where the reference to Weyl's equidistribution article quoted above, make sense. However, Weyl's article contains a deep method involving the estimation of trigonometric sums (also called complex exponential sums) that is still very important in number theory, whereas the reasons for the equidistribution of the sequence used in the method are quite simple.

A C++ impementation of a subroutine carrying out the key steps of this method is as follows:

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  uint32_t middle(uint64_t &x,uint64_t &w, uint64_t &z)
5  {
6    x *= x;
7    w += z;
8    x += w;
9    return x = (x>>32) | (x<<32);
10 }
```

We will later include a description of a calling program for this subroutine, a way to compile this program, and the output produced. The way this subroutine works can be explained independently, however, and a line-by-line explanation explanation follows. As before, the numbers at the beginning of the lines are line numbers, and they are not part of the file. For later reference, we will call this file `middle.cc`.[6.6] Line 1 is present in virtually all C or C++ programs, and the file `stdio.h` mentioned in this line contain the standard C input/output library. The file `stdint.h` in the second line contains the integer types mentioned on line 4. The types `uint32_t` and `uint64_t` unsigned integer types of the indicated size in bits. The operations on these numbers are done in modular arithmetic; that is, any digits of the result that cannot be stored are simply lost, without any error message. Thus, the square of a 64 bit integer would need 128 bits to store, and when a location of type `uint64_t` is used to store the result, the 64 bits on the left are lost. The subroutine is called by reference, that is, any of change in the variables `x`, `w`, and `z` will result in the same change of the calling variables; the best way to think about this is that the command calling the subroutine simply transfers the memory location of the calling variables to the subroutine, and the subroutine uses the same locations to do the calculation. Line 4 could also have been written as

```
uint32_t middle(uint64_t& x,uint64_t& w, uint64_t& z)
```

which indicates more clearly that, for example, `x` is a variable of type `uint64_t&` rather than the address `&x` is of type `uint64_t`. On the other hand, `&x` indicates a reference to `x`.

In the body of the subroutine, line 6 squares `x`; in C and C++ the assignment operator is simply the equation sign `=`. The equation `x *= x` is an abbreviation of `x = x * x`, where the asterisk tt * is the multiplication sign. This abbreviation is helpful for the compiler in that it tells the compiler to store the argument at the location of the first argument; the compiler would probably figure this out on its own, still abbreviations like this speed up compilation. Here `x` being a 64 bit unsigned integer, its square is 128 bits, but the 64 bits on the left are discarded. On line 8, the operation `w = w + z` is performed, again abbreviated in the same style as the abbreviation used on line 7. The content of `z` will stay the same throughout, while `w` will contain the terms of the equidistributed sequence. The starting values of `x`, `w`, and `z` are the seeds of the random number generation. There are no special requirements on the first two of these; it is important, however, that `z` be odd and that the upper 32 bits of `z` represent a nonzero number.

On line 8 `w` is added to `x`. On line 9, the expression `x>>32` indicates shifting the digits of `x` to the right; during the shift, 32 zeros from the right will enter the location, so the result of this shift contains 32 zeros on the left, and the lower 32 digits of the middle 64 digits of the square of

[6.6] The compiler `gcc` that we will use for compiling this program originally used the `*.cc` file extension for C++ programs. Nowadays it also recognizes other file extensions, such as `*.cpp`. The name `gcc` used to stand for GNU C compiler, but it has been renamed GNU Compiler Collection, since it can compile a number of languages other than C.

the original value of `x`, as modified by the addition of `w`. These lower 32 digits are returned by the subroutine on line 9. On this line, `x<<32` indicates shifting the value (calculated on line 8 of `x` by 32 digits to the left. On the right, 32 zeros will enter as a result of the shift, and the lower 32 digits of the value of `x` will now occupy the upper 32 digits. The symbol `|` on this line indicates inclusive or (`0 | 0 = 0`, `1 | 0 = 0`, `0 | 1 = 0`, and `1 | 1 = 1`), so the result of this operation is that the upper and the lower 32 digits of `x` change places. This new value of `x` will be the new value of the corresponding calling parameter, but the subroutine will only return the lower 32 bits, since the subroutine was declared `uint32_t` type.

The program doing the calculations consists of three files placed in the same directory. The subroutine `middle.cc` already mentioned, the header file `middle.h` and the calling program `main.cc`. The header file `middle.h` consists of a single line

```
1  uint32_t middle(uint64_t &x,uint64_t &w, uint64_t &z);
```

The number at the beginning of the line is not part of the file – it is a line number. Admittedly, it is not important to indicate it here, but we included it for uniform treatment of file listings. This line declares the subroutine used in the program, which needs to be done before the main program; notice that the declaration here is followed by a semicolon, while it is not followed by one in the file `middle.cc` given above. There was no compelling reason to place it in a separate file, we mainly did this to indicate how this can be done, and to point out some differences between C and C++ below. The listing of the calling program `main.cc` is as follows (it is not required to call this file `main.cc`):

```
1  #include <stdio.h>
2  #include <stdint.h>
3  #include "middle.h"
4
5  main()
6  {
7    FILE *inp, *outp;
8    uint64_t y, u, s;
9    uint32_t rand;
10   double randf;
11   int i;
12   inp = fopen("seeds", "r");
13   outp = fopen("randoms", "w");
14   fscanf(inp, "%llx\n%llx\n%llx\n", &y, &u, &s);
15   for (i=0; i < 6; i++) {
16     rand = middle(y,u,s);
17     fprintf(outp,"hexadedimal:    y=%016llx\n", y);
18     fprintf(outp,"hexadecimal: rand=        %08x\n", rand);
19     fprintf(outp,"rand =%ld\n", rand);
20     randf = rand;
21     fprintf(outp,"randf=%f\n", randf);
22   }
23   fclose(inp);
24   fclose(outp);
25 }
```

The first to `#include` lines are the same as in the file `middle.cc` on page 23 above. Line 3 includes the header file `middle.h` here; equivalently, the content of the file could be included here (but there are some subtle differences in how the program is compiled – see below). Note the differences in syntax: On lines 1 and 2, the less than and greater than signs `<` and `>` indicate that the mentioned

files are stored at the standard location for all include files, whereas on line 3, between the quotes the full path to the file needs to be given (at present, this is just the name of the file, since the program `main.cc` is in the same directory as the header file `middle.h`.

On line 5, the name `main` is the required name of the main program. On line 7 the file pointers `*inp` and `*outp` are declared. On line 8, the 64 bit unsigned integers are declared; initially these will be given the seed values. On line 9 the 32 bit unsigned integer `rand` declared On line 10, the double precision floating number `randf` is declared. This is because the calculated random numbers are most likely used as floating point numbers. On line 11, the integer `i` is declared; it will be used as a loop variable to produce a sequence of random numbers. On line the file `seeds` with file pointer `inp` is opened for reading, and on line 13, the file `randoms` with file pointer `outp` is opened for writing; this is what the quoted letters `r` and `w` indicate. Line 14 reads the variables `y`, `u`, and `s`, are read from the input file `seeds` (the file is indicated by the file pointer `inp` identifies the file. The variables need to be stated as references in the function `fscanf` (meaning read from file); the second argument of this function is the format string, which identifies the numbers as long hexadecimal integers (`%llx` in the format string), and the numbers are separated by newline charachters `\n` in the input line (i.e., they are placed on separate lines). This means that the inputs are required to be long long (meaning 64 bit) hexadecimal numbers; 4 bits correspond to a hexadecimal digit, so the numbers need to be 16 digit hexadecimal numbers. There are no special requirements on the seeds except that `s` must be an odd number (so its rightmost bit must be `1`), and its upper 32 bits should represent a nonzero numbers. There are conversion issues in making sure that these conditions are satisfied in case `s` is read as a decimal integer; these could be overcome by some additional complications in the program, but we did not want to deal with this issue. Here is the input file `seeds`:

```
1  c305f794a1b2c904
2  b3c9125ef907b352
3  a3569b02c4d3b9a5
```

The numbers are line numbers.

On line 16 the subroutine `middle` is called. It returns the random number and updaes the values of `y`, `u`, and `s`. Line 17 prints `y` to the output file `randoms` as a hexadecimal 16 digit hexadecimal number with leading zeros included (this is what is indicated by `016` in the format string `%016llx` in the format string; `x` indicates that the output is hexadecimal, and `ll` indicates that a *long long* integer is to be printed to the same file. On line 19 the integer `rand` is printed as an 8 digit hexadecimal integer. On line 20, the integer `rand` is converted to the double floating point number `randf`, and on line 21 this number is printed to the output file. The file pointers `inp` and `outp` close the files `seeds` and `randoms` corresponding to these file pointers on lines 23 and 24. Here is the output file `randoms`:

```
 1  hexadedimal:    y=ed42b507a6082fb4
 2  hexadecimal: rand=        a6082fb4
 3  rand =2785554356
 4  randf=2785554356.000000
 5  hexadedimal:    y=f6d2bd2ceedc0a5c
 6  hexadecimal: rand=        eedc0a5c
 7  rand =4007397980
 8  randf=4007397980.000000
 9  hexadedimal:    y=260e31514e8a2b6c
10  hexadecimal: rand=        4e8a2b6c
11  rand =1317677932
12  randf=1317677932.000000
13  hexadedimal:    y=b4240f76c11acdca
```

```
14 hexadecimal: rand=        c11acdca
15 rand =3239759306
16 randf=3239759306.000000
17 hexadedimal:    y=d79f76ef8fc6e8bd
18 hexadecimal: rand=        8fc6e8bd
19 rand =2412177597
20 randf=2412177597.000000
21 hexadedimal:    y=93f128b93d4b53e5
22 hexadecimal: rand=        3d4b53e5
23 rand =1028346853
24 randf=1028346853.000000
```

These show that the integer `rand` is just the last eight hexadecimal digits (i.e., 32 bits of the integer `y` on the preceding line; why this is so is explained in the discussion of the subroutine. The printout also show that the integer value of `rand` and the double floating point value of `randf` agree. This is important, especially as far as the printouts on lines 7 and 8 are concerned, since the integer printed on line 7 is greater than 2^{31}; in fact, $2^{31} = 2147483648$ and $2^{32} = 4294967296$. Since calculation with integers of `uint32_t` are performed modulo 2^{32}, it is conceivable that integers between 2^{31} and 2^{32} would be converted to negative numbers.

To compile these programs, it is convenient to use what is called a makefile, which should be a file with the name `makefile` or `Makefile`. We used the following file called `makefile`:

```
1 all : middle
2 middle : middle.o main.o
3         gcc -o middle -s -O4 middle.o main.o -lm
4 middle.o : middle.cc
5         gcc -c -O4 middle.cc
6 main.o : main.cc middle.h
7         gcc -c -O4 main.cc
```

A very important point not visible in the listing of this file is that on lines 3, 5, and 7 the initial white space must be a single tab character (and not a number of space characters). On the left-and side of the colon `:` are the targets; there are often names of files to be created, but more generally, they are goals to be accomplished; for example, there will be no file named `all` mentioned on line 1. On the right-hand side of the colon are the files necessary to accomplish this task, and in the next line the commands used to accomplish the task; we will give an explanation of the meanings of these commands somewhat later. This helps one in performing two functions. First, one does not need to remember each time how to compile the program; second, only those task will be carried out where the target has an earlier date than the dates of the files on the right-hand side of the colon. To run a `makefile`, one needs to type

```
$ make all
```

(here the dollar sign `$` is a prompt, and must not be typed). More generally, to perform a specific task, `all` should be replaced with the name of the task. One can often type `make` instead of `make all`. Only those tasks will be performed where the target is out of date, that is the files on the left-hand side are newer than the files on the right-hand side; that is, if the file `middle` is not out of date then typing `make all` will do nothing. As seen from the make file, the name of the compiled program is `middle`; to run the program, we need to type its name on the command line. That is, we need to type

```
$ middle
```

Line 6 of the `makefile` requires a special command. The compilation would work even if the name of the header file `middle.h` does not appear on line 6, yet there is a problem. If we successfully compiled `middle` and then change the file `middle.h`, the only reason the compilation is performed again is that `middle.h` appears on line 6.

As far as the commands on lines 5, and 7 of the `makefile`, the compiler command `gcc` is used with the `-c` option to produce the `*.o` files containing the compilation. The option `-04` specifies the level of compiler optimization to be used. On line 3 the command `gcc` links the `*.o` files containing the compilations listed after the option `-s`; the optimization level `-04` is also specified; the mathematics library, indicated by `-lm` at the end of the line. The option `-o` specifies the name `middle` of the linked program. There are many options that can be used with the `gcc` compiler; these are described in the document Using the GNU Compiler Collection.

6.3 The updated middle square method: a C implementation

We will describe how to implement the same method using C instead of C++. The subroutine was put in the file `middle.c`, which is as follows:

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  uint32_t middle(uint64_t *x, uint64_t *w, uint64_t *z)
5  {
6    *x *= *x;
7    *x += (*w += *z);
8    return *x = (*x>>32) | (*x<<32);
9  }
```

Comparing this to the C++ version of the program `middle.cc` on p. 23, the first difference is in the naming of the file, to indicate that the present one is a C program. Aside from this, the main difference is that here we use pointers and the quantities manipulated are the targets of the pointers, indicated by a preceding asterisk `*`, whereas in the C++ version, only the function declaration (line 4 in the C++ version) mentions references, the body mentions the variables themselves. The function declaration in the C version could have also been written as

```
uint32_t middle(uint64_t* x, uint64_t* w, uint64_t* z)
```

This form indicates more clearly that, for example, the variable `x` is declared as a pointer. The idea here is the same as in the C++ version, where we were also able to rewrite the function declaration. The difference between line 7 here and lines 7 and 8 in the C++ version is inessential, because even there lines 7 and 8 could have been combined into a single line as

```
x += (w += z);
```

but when we explained the C++ version, we did not want to complicate our explanation with an unimportant contraction of two lines into a single line. The effect of this contracted line is exactly the same as the two lines mentioned in the C++ version of the program.

The next file is the header file `middle.h` is as follows.

```
1  uint32_t middle(uint64_t *x, uint64_t *w, uint64_t *z);
```

Comparing this to the C++ version of the file on p. 24, the name of the file `middle.h` is the same as above; the difference is only that the declaration of the subroutine is different in C from that is C++.

The next file called `main.c` contains the calling program (it is not requied to call this file `main.c`:

```
1  #include <stdio.h>
2  #include <stdint.h>
3  #include "middle.h"
4
5  main()
6  {
7    FILE *inp, *outp;
8    uint64_t y, u, s;
9    uint32_t rand;
10   double randf;
11   int i;
12   inp = fopen("seeds", "r");
13   outp = fopen("randoms", "w");
14   fscanf(inp, "%llx\n%llx\n%llx\n", &y, &u, &s);
15   for (i=0; i < 6; i++) {
16     rand = middle(&y,&u,&s);
17     fprintf(outp,"rand =%ld\n", rand);
18     randf = rand;
19     fprintf(outp,"randf=%f\n", randf);
20   }
21   fclose(inp);
22   fclose(outp);
23 }
```

The only difference here is the way the subroutine is called on line 16, since here the addresses of the variables are listed as arguments. Other than this, we did not print out the hexadecimal versions of the variables `y` and `rand`, since earlier we printed them out only to explain how the program worked. The printout of the program is put in the file `randoms`, which is as follows:

```
1  rand =2785554356
2  randf=2785554356.000000
3  rand =4007397980
4  randf=4007397980.000000
5  rand =1317677932
6  randf=1317677932.000000
7  rand =3239759306
8  randf=3239759306.000000
9  rand =2412177597
10 randf=2412177597.000000
11 rand =1028346853
12 randf=1028346853.000000
```

The difference here is that the the hexadecimal numbers are missing, but the decimal numbers themselves are exactly the same as before.

We will discuss the file `makefile` next:

```
1  all : middle
2  middle : middle.o main.o
3           gcc -o middle -s -O4 middle.o main.o -lm
4  middle.o : middle.c
5           gcc -c -O4 middle.c
6  main.o : main.c middle.h
7           gcc -c -O4 main.c
```

The only difference here is that the file name extensions are `.c` for the C programs and `.cc` for the C++ programs.

7 Gasoline inventory model

We will discuss a maxima implementation of a gasoline intentory model described in [1, Section 5.4, pp 203–210]. In our implementation, every five days a fixed amount of gasoline is delivered to a gasoline station. The amount of daily purchases is random, but the distribution of the random variable describing these purchases is experimentally known. In the model, we use cubic splines, described next, to represent the inverse of the distrubution of this random variable. Then we consider the cost incurred by the deliveries, and the daily amount of gasoline remaining. The issue is to set the amount of gasoline deliveries to a level that so that the station is unlikely to run out of gasoline. At the same time the cost of running the gas station needs to be as small as feasible. There are several variables that can be changed in the model, such as the frequency of deliveries and the amount delivered.

7.1 Cubic splines

Let $[a, b]$ be an interval, and assume we are given points x_0, x_1, ..., x_n with $a = x_0 < x_1 < \ldots < x_n = b$ for some positive integer n. and corresponding values y_0, y_1, ..., y_n, we would like to determine cubic polynomials

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

for i with $0 \le i < n$ satisfying the following conditions:

$$\begin{aligned} S_i(x_i) &= y_i \quad \text{for } i \text{ with } 0 \le i \le n-1, \\ S_{n-1}(x_n) &= y_n, \\ S_i^{(k)}(x_{i+1}) &= S_{i+1}^{(k)}(x_{i+1}) \qquad \text{for } i \text{ with } 0 \le i \le n-2 \text{ and } 0 \le k \le 2. \end{aligned}$$

These represent $(n+1)+3(n-1) = 4n-2$ equations. The two remaining equations can be obtained by setting conditions on S_0 at x_0 and S_{n-1} at x_n. For example, if one wants to approximate a function f with $f(x_i) = y_i$, then one can require that

$$S_0'(x_0) = f'(x_0) \quad \text{and} \quad S_{n-1}'(x_n) = f'(x_n);$$

these are called *correct boundary conditions*. The drawback of these conditions is that $f'(x_0)$ and $f'(x_n)$ may not be known. Alternatively, one may require that

$$S_0''(x_0) = 0 \quad \text{and} \quad S_{n-1}''(x_n) = 0;$$

these are called *free* or *natural boundary conditions*. The function $S(x)$ defined as

$$S(x) = S_i(x) \quad \text{whenever} \quad x \in [x_i, i+1] \qquad (1 \le i < n)$$

is called a *cubic spline*. The name spline is that of a flexible ruler forced to match a certain shape by clamping it at certain points, and used to draw curves. These equations are not hard to solve on computers. See [2, Section 37, p. 167–174] for details.

7.2 The maxima model

We will explain the details of the model as we describe the maxima script implementing the model. Here is the script:

```
1  kill(all);
2  linel : 60;
3  load(interpol)$
4  pairs : [[0,1000], [0.015,1050], [.03, 1150], [0.07, 1250],
5     [.2, 1350], [.4,1450], [.68, 1550],
6     [.83, 1750], [.93, 1860], [1.0, 1950]]$
7  inv_distr_spline : cspline(pairs)$
8  inv_distr(x) :=  ''inv_distr_spline$
9  st : make_random_state(37098341348)$
10 set_random_state(st);
11 time : 5;
12 deliv : 7200;
13 inventory : 0;
14 days : 40;
15 cost : 0;
16 delivcost : 2000;
17 remaining_days : days;
18 storage_cost : .5;
19 flag : 0;
20 daily_amts : [];
21 for k : 0 step 0 while flag = 0 do(
22   inventory : inventory + deliv,
23   cost : cost + delivcost,
24   if time >= remaining_days then (time : remaining_days,
25         flag : 1),
26   for i : 1 step 1 thru time do(
27      consump : float(inv_distr(random(1e8)/1e8)),
28      inventory : inventory - consump,
29      if inventory < 0 then inventory : 0
30         else cost : cost + inventory * storage_cost,
31      remaining_days : remaining_days - 1,
32      dayno : days - remaining_days,
33      daily_amts : append(daily_amts,[[dayno,inventory]])
34    )
35 );
36 delete_file(mkp("gasoline.output"));
37 writefile("gasoline.output");
38 daily_amts;
39 avgcost : float(cost/days);
40 closefile();
```

We will explain what this script does line by line. We will not discuss the output, since the output contains a lot of unimportant information; in fact, it contains 128 line, much of it is no interest except while debugging the script. Instead, we print a part of the output to a file, and we will discuss that file. On line 3 we load the maxima package interpol discussing interpolation. On lines 4–6, the ordered pairs listed describe the experimental distribution of the random variable representing the daily gasoline purchases. Calling this random variable X, the fourth pair on this list is `[.2, 1350]`; this means that $P(X \leq 1350) = .2$, where the amount of gasoline 1350 is in gallons. The other pairs

are interpreted similarly. The function $f(x)$ that for the first member of the pair gives as value the second member of the pair in the inverse of the distribution function of X. We are only given this function at specific points; the function is extended to all points in the interval $[0, 1]$ using cubic splines. On line 7 this spline is defined as `inv_distr_spline`; this object is a maxima *noun*; on line 8, the operator `''` (two single quotes) converts this maxima noun to a maxima it verb; maxima functions must be verbs. On line 8, `x` is the default variable of the expression `''inv_distr_spline` (it is possible to choose a different variable name). Thus the function `inv_distr` on the left is assigned its values by using the variable `x`. Line 9 create a random state, and on line 10 this state is assigned to the random number generator. On lines 11 through 18 the variables of the model are initialized; `time` is the number of days between deliveries, `deliv` is the amount of gasoline deliveredd in gallons, `inventory` is the amount of gasoline held, `days` is the number of days the model runs for, `cost` is the cost so far incurred, and `delivcost` is the cost of each delivery, `remaining_days` will hold the number of days left for the model to run, and `storage_cost` is the cost of holding gasoline on inventory in dollars per gallonr. The variable `flag` will be set to `1` when the model has run the number of days it supposed to run for. The object `daily_amts` will hold the ordered pairs consisting of the day number and the amount of of gasoline on inventory at the end of the day. On line 21 a *while* loop is started; it is written as a *for* loop, but the loop variable `k` plays no role. In the loop the cmmands are separated by commas.

The loop starts with a delivery of gasoline on line 22; on line 23 the cost of delivery is added to the cost. On lines 24–25, if `time` (the time between deliveries) is greater than the remaining days in the model, `flag` is set to `1`. On line 26, the loop describing the days to the delivery is started. On line 27, the consumption is evaluated as the value of the inverse distrubution value at a random number between 0 and 1. On line 28, the inventory decreased by the amount of consuption. In lines 29–30, if the inventory were to become negative if the consumption amount is subtracted, then the inventory is set to zero. This means that all gasoline was sold out; the inventory of course will not become negative. If the inventory is positive, the storage cost for the day is added to the cost on linw 30. The number of remaining days is decremented on line 31; the day number is determined on line 32. On line 33m the pair consisting of the day number and the inventory is added to the list of daily amounts.

On lines 36–40, the list of the daily amounts is written to a file called `gasoline.output`. On line 36, this file is deleted; this is important, because if the script was run earlier, the file would contain the earlier output; if one does not want to preserve this output, the file first needs to be deleted (the function `mkp` on this line stands for "make path" – what this means is unimportant for our purposed at present. On line 37, the command tells maxima to write the output of the lines following also to the file `gasoline.output`. On line 37 the daily amounts are written out (this is the list of pairs created in the loop). On line, the average daily cost is calculated. On line 40, the file `gasoline.output` is closed (the file does not need to be named here, since Maxima remembers which file it opened).

The content of the file `gasoline.output` follows. This is mostly self-expanatory. The file was modified in that some lines were too long for this manuscript, and they were broken up.

```
1  Starts dribbling to gasoline.output (2018/4/21, 18:21:25).
2  NIL
3  (%o20)                       done
4  (%i21)                    daily_amts
5  (%o21) [[1, 5835.075835622294], [2, 4615.108362796575],
6  [3, 2827.917683564751], [4, 1367.206106675059], [5, 0],
7  [6, 5734.867178081771], [7, 4402.764396916923],
8  [8, 2941.238886947169], [9, 1689.35860564159],
```

```
 9  [10, 194.9088484215324], [11, 5569.681852841805],
10  [12, 4146.742630750356], [13, 2738.477853429898],
11  [14, 1715.617354278904], [15, 53.54455238216633],
12  [16, 5930.062047276927], [17, 4595.26267538915],
13  [18, 3312.704960077897], [19, 1550.610891136599],
14  [20, 140.6295407520802], [21, 6097.536562303907],
15  [22, 4659.742001560235], [23, 3290.925973632516],
16  [24, 1490.945197194474], [25, 0], [26, 5742.492665305812],
17  [27, 4072.7906042983], [28, 2929.058232127631],
18  [29, 1485.686624854492], [30, 28.2759371630591],
19  [31, 5985.605430196491], [32, 4249.946879513068],
20  [33, 2622.787234154956], [34, 1370.380496369743], [35, 0],
21  [36, 5393.9207154952], [37, 3934.923195257497],
22  [38, 2553.880712494067], [39, 940.9412032508558], [40, 0]]
23                                        cost
24  (%i22)             avgcost : float(----)
25                                        days
26  (%o22)               1852.645249101947
27  (%i23)                  closefile()
```

8 Queueing models

8.1 The subject of queueing theory

In queueing theory,[8.1] one studies waiting on lines mathematically. Clients arrive randomly, to be served by a number of servers. Until a server is available, the clients may wait on a number of lines, or abandon being served altogether. The length to serve a client may also be random, and it may depend on the kind of service to be performed. The order in which the clients are served may also vary according to various schemes: first come, first served (FIFO, for First In, First Out), last come, first served (LIFO, for Last In, First Out). There are many real world situations that can be described this way, from supermarket checkout lines to telephone exchanges, and computer CPUs (Central Processing Units) in a multiprocessing operating system.

In fact, multiprocessing operating systems in computers represent an interesting example. Here the servers are the CPUs (a computer may have several CPUs), the clients are the programs (processes) to be executed. One CPU often serves several clients; before finishing a program, the CPU may turn to do a part of another program. Serving some clients is never finished (some programs or processes always run when the computer is on).

While some queueing situations can be handled theoretically, in other cases modeling is a better approach.

8.2 A model of ships unloading in a harbor

In this subsection we will implement the queueing model of ships waiting in a harbor for unloading, using Maxima. The model is described in [1, Section 5.5, pp. 213–219].In a harbor, ships arrive randomly. Only one ship can be unloaded at a time, the other ships have to wait. We changed the parameters of the model somewhat to what was given in the quoted book. The Maxima script called

[8.1] The spelling "queueing" is used in the literature on *queueing theory*. In ordinary usage, the spelling "queuing" is more common. In fact, the book [1, Section 5.5, p. 213] uses the latter spelling.

`harbor.max` implementing the model is given next. Below the script, we will outline how this script works, along with a closer description of the model.

```
 1  kill(all);
 2  linel : 60;
 3  ships : 100;
 4  betwmin : 15;
 5  betwmax : 140;
 6  unlmin : 45;
 7  unlmax : 95;
 8  invdist_betw(x) := betwmin+(betwmax-betwmin)*x;
 9  invdist_unld(x) := unlmin+(unlmax-unlmin)*x;
10  arrive[0] : 0;
11  finish[0] : 0;
12  unload[0] : 0;
13  hartime : 0;
14  maxhar : 0;
15  maxwait : 0;
16  waittime : 0;
17  st : make_random_state(17253062308)$
18  set_random_state(st);
19  for i : 1 step 1 thru ships do(
20    between[i] : invdist_betw(random(1e8)/1e8),
21    unload[i] : invdist_unld(random(1e8)/1e8),
22    arrive[i] : arrive[i-1] + between[i],
23    timediff : arrive[i]-finish[i-1],
24    if timediff >= 0 then (idle[i] : timediff, wait[i] : 0)
25        else (wait[i] : -timediff, idle[i] : 0),
26    start[i] : arrive[i] + wait[i],
27    finish[i] : start[i] + unload[i],
28    harbor[i] : wait[i] + unload[i],
29    hartime : hartime + harbor[i],
30    if harbor[i] > maxhar then maxhar : harbor[i],
31    waittime : waittime + wait[i],
32    if wait[i] > maxwait then maxwait : wait [i]
33  );
34  avg_hartime : float(hartime/ships);
35  avg_waittime : float(waittime/ships);
36  idletime_fraction : float(idletime/finish[ships]);
37  delete_file(mkp("harbor.output"));
38  writefile("harbor.output");
39  avg_hartime;
40  maxhar;
41  avg_waittime;
42  maxwait;
43  idletime_fraction;
44  closefile();
```

We will not give the output of this program; the part of the output that is of interest to us we printed to a file. As before, the line numbers are not not part of the file containing the program, but they will be helpful in a line-by-line description of the program. Line 3 specifies that the service of 100 ships will be simulated. The time between the arrival of successive ships and the unloading time of each ship is described by random variables. The inverses of the distribution functions of these random

variables are given in lines 8 and 9. In the present model, these inverses are linear functions given in terms of the paramemters given values on lines 4–7: `betwmin` is the minimum time and `betwmax` is the maximum time between the arrivals of successive ships, `unlmin` is the minimum time and `unlmax` is the maximum time taken for the unloading of each ship. In necessary, more complicated distributions can be taken for modeling the arrivals and the unloading times of the ships. The data given are somewhat different from those given in the book: the time between successive arrivals of ships is at least 15 minutes and at most 140 minutes, the the arrival time is uniformly distributed between these values. The minimum time taken for unloading is 45 minutes, and the maximum time is 95 minutes, and between these limits the time taken for unloading is uniformly distributed.

There is no ship 0, the variables on lines 10–12 are initiated to be zero, so that the arrival times and the times for finishing the unloading of each ship can be calculated the same way for ship number 1 as for all other ships. The variable `hartime` will be the total time spent in the harbor by the ships, `maxhar`, the maximum time spent by a ship in the harbor, `maxwait`, the maximum time spent waiting by a ship until the start of her unloading, and `waittime`, the total time spent waiting for unloading by the ships. On line 17, a random state `st` is created, and on line 18, this random state is assigned to the random number generator.

The arrival and serving of each ship is described in the loop on lines 19–33. it is a `for` loop up to `ships`, which is the number of ships simulated in the model. On line 20 the time between the arrival of ships `i` and `i-1` is given as a random value of the inverse distribution function `invdist_betw`. Similarly, on line 21, the unloading time of ship `i` is obtained as a random value of the inverse distribution function `invist_unld`. On line 22 the arrival time of ship `i` is calculated. The difference, calculated on line 23, between the time of the arrival of the `i`th ship and the time the (`i-1`)st ship was unloaded decides whether the unloading of ship `i` can start immediately or whether the ship has to wait for a time called `wait[i]`. The start time of ship `i` is calculated on line 26, and the time the unloading of this ship is finished is given on line 27. On line 28 time time spent in the harbor by this ship is calculated. This time is added to `hartime`, which will be the sum of the times each ship spends in the harbor. On line 30, the variable `maxhar` contains the maximum value each ship prior to the `i`th spent in the harbor. If the `i`th ship spends more time than this, the `maxhar` will be given as value the time spent by this ship; otherwise, its value is unchanged. On line 31, the time spent waiting by ship `i` is added to the variable `waittime`, whose value at this point is the total time spent waiting by the earlier ships. On line 32, the maximum waiting time `maxwait` is updated, in a way similar to the updating of `maxhar` was done on line 30.

Outside the loop, the average time spent in the harbor and the average waiting time by a ship is calculated on lines 34–35, and on the line 36, the fraction of time spent in the harbor is divided by the total harbor time used is calculated as `idletime_fraction`. While the values of these variables are written to the output (not presented here) along with the formulas calculating them, in the end we are only interested in the values, and not how the model calculated them; hence we want to print them to a file. On line 37, the earlier version of the file `harbor.output` is deleted if it exists (it would exist if the present script is run several times); without deleting it, the output written to this file would be appended to its current content. On line 37, this file is opened for writing, and the values of the variables lsted on lines 39–43 are written to this file. The file is closed on line 44. The content of the file `harbor.output` is as follows:

```
1  Starts dribbling to harbor.output (2018/2/18, 22:35:2).
2  NIL
3  (%o23)                          done
4  (%i24)                     avg_hartime
5  (%o24)                 109.1153784722558
6  (%i25)                        maxhar
```

```
 7  (%o25)                247.1048460751318
 8  (%i26)                   avg_waittime
 9  (%o26)                40.27000180876873
10  (%i27)                     maxwait
11  (%o27)                176.8328431409964
12  (%i28)                idletime_fraction
13  (%o28)          1.274634864146866E-4 idletime
14  (%i29)                   closefile()
```

9 Stochastic matrices

Definition 9.1. A square matrix is called a *stochastic matrix* if all its entries are nonnegative, and all its columns sum up to 1. A *probability vector* is a column vector with nonnegative entries such that the sum of its entries is 1.

The intuitive interpretation of an $N \times N$ stochastic matrix $A = (a_{ij})$ is the following. A system has N states S_1, S_2, ..., S_N, and the evolution of the system is considered for discrete times $n = 0$, 1, 2, ... If at time step n the system is in state S_i, then the probability that the system will be in state S_j at time step $n+1$ is a_{ij}; these quantities are called *transition probabilities*, and the process described this way is called a *Markov process* with a constant transition matrix. Generalizations with a transition matrix changing at each step is possible, but they will not be considered at this point. If $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$ is the initial state probability vector, meaning that the system at time 0 is in state S_i with probability x_i, then the state probability vector describing the system at the nth time step is $A^n\mathbf{x}$.

Lemma 9.1. *Let $N > 0$ be an integer, and let A and B be $N \times N$ stochastic matrices. Then AB is also a stochastic matrix.*

Proof. Let the entry at in the ith row and j column in A be a_{ij}, in B, b_{ij} and in $C = AB$, c_{ij}. We clearly have $c_{ij} \geq 0$. Furthermore,

$$\sum_{i=1}^{N} c_{ik} = \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}b_{jk} = \sum_{j=1}^{N}\Big(\sum_{i=1}^{N} a_{ij}\Big)b_{jk} = \sum_{j=1}^{N} b_{jk} = 1,$$

where the third equality holds because each column sum of A is 1, and the fourth equality holds because each column sum of B is 1. This shows that each column sum of C is also 1. □

Lemma 9.2. *Let $N > 0$ be an integer, let A be an $N \times N$ stochastic matrix, and let $\mathbf{x}$ be a probability vector size N. Then $A\mathbf{x}$ is also a probability vector.*

The proof is essentially the same as that of the preceding lemma.

Proof. Let a_{ij} be the entries of the matrix A and let x_i be the entries of $\mathbf{x}$, and let y_i be the entries of $\mathbf{y} = A\mathbf{x}$. Then

$$\sum_{i=1}^{N} y_i = \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}x_j = \sum_{j=1}^{N}\Big(\sum_{i=1}^{N} a_{ij}\Big)x_j = \sum_{j=1}^{N} x_j = 1.$$

Since we clearly have $y_i \geq 0$, this shows that $\mathbf{y}$ is a probability vector. □

Definition 9.2. Let $N > 0$ be an integer, and let $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$ be a size N column vector. The l_1 norm of $\mathbf{x}$ is defined as

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{N} |x_i|.$$

It is easy to see that $\|\cdot\|$ is indeed a norm; in particular,

$$\|\lambda\mathbf{x}\|_1 = |\lambda|\,\|\mathbf{x}\|. \tag{9.1}$$

Corollary 9.1. *Let $N > 0$ be an integer, let A be an $N \times N$ stochastic matrix and let $\mathbf{x}$ be a column vector. Then $\|A\mathbf{x}\|_1 \le \|\mathbf{x}\|_1$.*

Proof. For any vector $\mathbf{z} = (z_1, z_2, \ldots, z_N)^T$ write

$$|\mathbf{z}| = (|z_1|, |z_2|, \ldots, |z_N|)^T.$$

Let $A = (a_{ij})$, $\mathbf{x} = (x_1, x_2, \ldots x_N)$, and let $\mathbf{y} = (y_1, y_2, \ldots, y_N)^T = A\mathbf{x}$. For i with $1 \le i \le N$ we have

$$|y_i| = \Big|\sum_{j=1}^{N} a_{ij}x_j\Big| \le \sum_{i=1}^{N} a_{ij}|x_j|.$$

Since $a_{ij} \ge 0$, there was no need to put it in absolute value. The right-hand side is the ith component of the vector $A|\mathbf{x}|$. Since we have $\|A|\mathbf{x}|\|_1 = \||\mathbf{x}|\|_1 = \|\mathbf{x}\|_1$,[9.1] it follows that $\|A\mathbf{x}\|_1 \le \|\mathbf{x}\|_1$. □

Lemma 9.3. *Let $N > 0$ be an integer, let A be an $n \times n$ stochastic matrix, and let λ be an eigenvalue of A. Then $|\lambda| \le 1$.*

Proof. Assume, on the contrary, that $\lambda > 1$ is an eigenvalue of A, and let $\mathbf{x}$ be the corresponding eigenvector. Then we have

$$\|A\mathbf{x}\|_1 = \|\lambda\mathbf{x}\|_1 = |\lambda|\|\mathbf{x}\|_1 > \|\mathbf{x}_1\|_1.$$

This contradicts the conclusion of Corollary 9.1. □

Lemma 9.4. *Let $N > 0$ be an integer, and let A be an $N \times N$ stochastic matrix. Then 1 is an eigenvalue of A.*

Proof. The eigenvalues of A and A^T are the same. This is because, writing I for the $N \times N$ identity matrix, the eigenvalues of the former are the zeros of the polynomial $\det(A - \lambda I)$, and the eigenvalues of the latter are the zeros of the polynomial $\det(A^T - \lambda I)$, and these two polynomials are the same. We will show that 1 is an eigenvalue of A^T with the eigenvector $\mathbf{v} = (1, 1, \ldots, 1)^T$ of size N. Indeed, writing $A = (a_{ij})$, the ith entry of $A^T\mathbf{v}$ is

$$\sum_{j=1}^{N} a_{ji} \cdot 1 = 1,$$

because the column sums of A are 1. □

[9.1] The first equation holds according to Lemma 9.2, since we may assume that $|\mathbf{x}|$ is a probability vector in view of equation (9.1).

9.1 Limiting distribution of a Markov process

We have the following

Theorem 9.1. *Let A be a stochastic matrix, and assume A has no zero entries (i.e., all its entries are positive). Then there is a probability vector $\mathbf{y}$ with all positive entries such that for any probability vector $\mathbf{x}$ be of the appropriate size we have*

$$\lim_{n\to\infty} A^n\mathbf{x} = \mathbf{y}.$$

The result says, in other word, that if in a finite-dimensional Markov process with a constant transition matrix with positive entries, then there is a single limiting probability distribution to which the probabilities of the states converge. The theorem is a consequence of a much more general theorem called the Perron–Frobenius theorem discussed in the next subsection.

10 The Perron–Frobenius theorem

Theorem 10.1. *Let A be a square matrix all whose entries are positive.*

(a) *Then A has a positive eigenvalue r such that r is larger than the absolute values of all other eigenvalues of A.*

(b) *There is an eigenvector associated with r with all positive entries.*

(c) *Disregarding scalar multiples, there is only one eigenvection associated with r.*

(d) *The Jordan subspace associated with the eigenvalue r is one-dimensional.*

(e) *A has no other eigenvector with nonnegative entries.*

There are more conclusions that can be drawn, and in fact the theorem can be stated even with more generality, but this suffices for our purposes. Clauses (c) and (d) together mean that the multiplicity of the eigenvalue r is 1 (i.e., that r is a simple root of the characteristic equation of A). Later, we will say more about how to prove this theorem, but for now we will outline how this theorem can be used to prove Theorem 9.1.

Let $N > 0$ be an integer such that the matrix A in the theorem has size $N \times N$. To simplify our considerations, assume that we have $r = 1$ for the largest eigenvalue described in Clause (a) above.[10.1] According to [3, Theorem 8.2, on 20]. the matrix A is similar to a block matrix

$$B = \begin{pmatrix} 1 & 0_{1\times N-1} \\ 0_{(N-1)\times 1} & C \end{pmatrix}, \tag{10.1}$$

where the zeros with the subscripts indicate matrices if the indicated size with all zero entries, where C has the same eigenvalues as r except for the eigenvalue r;[10.2] the matrix C itself is a direct sum of Jordan block matries. In fact, the entry 1 in the first row and the first column represent a 1×1 Jordan block matrix.

Instead of using the language of matrices, it is better to use the language of linear transformations, where the matrix A is the representation of a linear transformation T of the vector space $\mathbb{C}_{N,1}$ (the

[10.1] If this is not already the case, replace the matrix A with the matrix A/r, whose largest eigenvalue is 1. Hence, our observations concerning the case $r = 1$ can easily translated back to the case of arbitrary r.

[10.2] The concepts used, such as similar matrices are all defined in the quoted notes [3]. Of course, A and B being similar, they must have the same eigenvalues.

space of size N column vectors, i.e., $N \times 1$ matrices over the field $\mathbf{C}$ of complex numbers into itself[10.3] in the canonical basis of unit column vectors $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^T$, $\mathbf{e}_2 = (0, 2, 0, \ldots, 0)^T$, $\mathbf{e}_3 = (0, 0, 1, \ldots, 0)^T$, ..., $\mathbf{e}_3 = (0, 0, 0, \ldots, 1)^T$. B represents the same linar tranformation in a different basis.[10.4] Writing $U = \mathbb{C}_{N,1}$ in the language of matrices, the space $\mathbb{C}_{N,1}$ on which T operates splits into the direct sum of two subspaces V and W, where V is the one-dimensional subspace spanned by an eigenvector $\mathbf{v}$ of A correspondig to the eigenvalue 1. The subspaces V and W are invariant subspaces of T. The transformation T restricted to V is the identity tranformation on V (since we assumed that $r = 1$, and all eigenvalues of $T \restriction W$, the restriction of T to W have absolute value less that 1. All eigenvalues of $T \restriction W$ have absolute value < 1; i.e., we have $\rho(T \restriction W) < 1$ ($\rho(\cdot)$ stands for spectral radius). Thus

$$\lim_{n\to\infty} T^n \mathbf{w} = 0 \qquad \text{for all} \quad w \in W \tag{10.2}$$

in view of Gelfand's Theorem 4.2.

Any vector in $\mathbf{u} \in U$ can be written in the form $\mathbf{u} = \alpha\mathbf{v} + \mathbf{w}$, where $\alpha \in \mathbb{C}$ and $\mathbf{w} \in W$. Since $T\mathbf{v} = \mathbf{v}$ (as $\mathbf{v}$ is an eigenvector of T corresponding to the eigenvalue 1), we have

$$\lim_{n\to\infty} T^n \mathbf{u} = \alpha\mathbf{v}. \tag{10.3}$$

These considerations allow us to prove Theorem 9.1.

The proof of Theorem 9.1. According to Theorem 10.1 Clause (a), there is a probability vector $\mathbf{u}$ that is an eigenvector of A corresponging to its eigenvalue 1. According to equation (10.2), for every probability vector $\mathbf{x}$, $\lim_{n\to\infty} A^n\mathbf{x} = \alpha\mathbf{u}$. As $A^n\mathbf{x}$ is a probability vector for all $n \geq 0$, their limit is also a probability vector; hence $\alpha = 1$. This establishes the result with $\mathbf{y} = \mathbf{u}$. □

10.1 The infinity norm

Given a size N column vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$, its l_∞ norm is defined as

$$\|\mathbf{x}\|_\infty = \max\{|x_k| : 1 \leq k \leq N\}.$$

Given a vector norm $\|\cdot\|$, one can associate with it a matrix norm called the the matrix norm *induced* by the vector norm $\|\cdot\|$ as follows. For a square matrix A we have

$$\|A\| = \sup\{\|A\mathbf{x}\| : \|\mathbf{x}\| = 1\},$$

where $\mathbf{x}$ runs over all column vectors of the appropriate size. For an induced matrix norm, we always have

$$\|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\| \tag{10.4}$$

and

$$\|AB\| \leq \|A\|\|B\|; \tag{10.5}$$

[10.3] See [3, Section 8, on p. 16] on representations linear transormations by matrices, similarity tranformations, Jordan canonical form, etc.

[10.4] As explained in [3] in the section mentioned, a similarity transformation of a matrix corresponds to a changed bases.

a norm satisfying this latter property, whether an induced norm or not, is called *submultiplicative*. The matrix norm $\|A\|_\infty$ induced by the vector norm $\|\mathbf{x}\|_\infty$ is easy to describe.Given an $N \times N$ matrix $A = (a_{ij})$ $(N > 0)$, we have

$$\|A\|_\infty = \max\Big\{\sum_{j=1}^{N} |a_{ij}| : 1 \le i \le N\Big\}, \tag{10.6}$$

that is, $\|A\|_\infty$ is the largest row sum of the absolute values of its entries. Indeed, $[A\mathbf{x}]_i$ for the ith entry of $A\mathbf{x}$, we have

$$\begin{aligned}\|A\mathbf{x}\|_\infty &= \max\big\{\big|[A\mathbf{x}]_i\big| : 1 \le i \le N\big\} = \max\left\{\Big|\sum_{j=1}^{N} a_{ij}x_j\Big| : 1 \le i \le N\right\} \\ &\le \max\Big\{\sum_{j=1}^{N} |a_{ij}| : 1 \le i \le N\Big\}\ \max\{|x_l| : 1 \le l \le N\} \\ &= \max\Big\{\sum_{j=1}^{N} |a_{ij}| : 1 \le i \le N\Big\}\|\mathbf{x}\|_\infty.\end{aligned}$$

If the largest absolute row sum is realized for row k $(1 \le k \le n)$, then equality holds here if $x_j = \overline{a_{kj}}/|a_{kj}|$, where $\bar{z}$ indicates the complex conjugate of z; we have $\|\mathbf{x}\|_\infty = 1$ for this choice. Equation (10.6) implies that if A and B are two $N \times N$ matrices size such that $|b_{ij}| \le |a_{ij}|$ for all i and j between 1 and N, then

$$\|B\|_\infty \le \|A\|_\infty. \tag{10.7}$$

Given a real p with $1 \le p < \infty$, the l_p-norm of a column vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$ is defined as

$$\|\mathbf{x}\| = \Big(\sum_{k=1}^{n} x_k^p\Big)^{1/p}.$$

The l_∞ norm is obtained by taking the limit when $p \to \infty$. So far, we used the l_1 and the l_∞ norms. The l_2 norm is also useful often, since it is the only one among these norms that is induced by an inner product. To show that these are norms, one needs to prove the Minkowski inequality $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$. This requires some effort except for the really easy cases of $p = 1$ and $p = \infty$. The case $0 < p < 1$ is also useful, but in this case the quantity $\|\cdot\|_p$ does not satisfy the Minkowski inequality, so the quantity in this case is not a norm: it is called a a *quasi norm*.

10.2 Proof of Clause (a) of the Perron–Frobenius Theorem

Proof of Clause (a) of Theorem 10.1. Since A is not the zero matrix, it has a nonzero eigenvalue, i.e., its spectral radius $\rho(A)$ is positive. Replacing the matrix A with the matrix $A/\rho(A)$, we may assume that the larges eigenvalue of A has absolute value 1. Assume that A has an eigenvalue λ with $\lambda \ne 1$ and $|\lambda| = 1$. Let k be a positive integer such that λ^k has a negative real part; the matrix A^k has all positive entries, and λ^k is one of its eigenvalues. Let $\epsilon > 0$ be smaller than all the diagonal elements of A^k. Then $\lambda^k - \epsilon$ is an eigenvalue lf $A^k - \epsilon I$, where I is the identity matrix for A.[10.5]

[10.5] I.e., it is the same size as A

Since $|\lambda^k - \epsilon| > 1$, this means that $\rho(A^k - \epsilon) > 1$. On the other hand, all entries of $A^k - \epsilon I$ are less than or equal to the corresponding entries of A^k, and they are all positive. Hence it follows that for every positive integer n, all entries of $(A^k - \epsilon I)^n$ are positive and they are less than or equal of the corresponding entries of A^n.[10.6] So, for all positive n, $\|(A^k - \epsilon I)^n\|_\infty \le \|(A^k)^n\|_\infty$ according to inequality (10.7). Hence $\rho(A^k - \epsilon I) \le \rho(A^k) = 1$, according to Theorem 4.2 (Gelfand's formula). This contradiction establishes the result. □

11 The power method for eigenvalues

Lemma 11.1. *Let A be a square matrix, and let $\lambda \neq 0$ be such that λ is an eigenvalue of A, and for any other eigenvalues of σ of A we have $|\sigma| < |\lambda|$. Let $\epsilon > 0$ and let $\mathbf{x}$ be a vector of an appropriate size. Then there is a vector $\mathbf{y}$ and an eigenvector $\mathbf{z}$ of A associated with the eigenvalue λ such that $\|\mathbf{x} - \mathbf{y}\|_\infty < \epsilon$ and there are nonzero numbers α_n such that*

$$\lim_{n\to\infty} \frac{\alpha_n}{\alpha_{n+1}} = \lambda. \tag{11.1}$$

and

$$\lim_{n\to\infty} \alpha_n A^n \mathbf{y} = \mathbf{z} \neq 0. \tag{11.2}$$

Further, $\mathbf{z}$ is an eigenvector of A associated with the eigenvalue λ. If $\lambda > 0$, we can choose $\alpha_n > 0$.

Once we prove this lemma, it is obvious that the statement is valid if we choose α_n to be the reciprocal of the entry of $A^n\mathbf{y}$ with the largest absolute value; if there are several of these, we can choose the reciprocal of the first one among them. This allows one to use the power method in practical calculations. Making this choice before the proof is complete would complicate the proof, so in the formulation of the lemma we did not do so. The content of this lemma is the main ingredient of the *power method* used to find an eigenvalue and eigenvector of a matrix – see [2, Section 30, pp. 184–196]; here we are interested in the lemma for theoretical reasons, so we are not going to discuss the numerical analysis aspect.

Proof for a Jordan block. To begin with, we will assume that A is a single Jordan block. After establishing the result for Jordan blocks, it will be easy to extend it to arbitrary matrices. A Jordan block is a $K \times K$ matrix of form

$$J = \sigma I + Z = \begin{pmatrix} \sigma & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \sigma & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \sigma & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & \sigma & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \sigma & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & \sigma \end{pmatrix}, \tag{11.3}$$

see [3, equation (8.9)]; here I is the $K \times K$ unit vector. Write $\mathbf{e}_k = (\delta_{k1}, \delta_{k2}, \dots, \delta_{kn})^T$ for the kth unit column vector, and write $\mathbf{e}_0 = 0$ Then we have

$$Z\mathbf{e}_k = \mathbf{e}_{k-1} \qquad (1 \le k \le K). \tag{11.4}$$

[10.6] This is easy to prove by induction and by using the following simple observation. If A_1, A_2, B_1, and B_2 are $N \times N$ matrices with all positive entries, and all entries of A_1 are less than or equal to the corresponding entries of B_1, and similarly for A_2 and B_2, then all entries of A_1A_2 are less than or equal to the corresponding entries of B_1B_2.

This means that $Z^n = 0$ for $n \geq K$.[11.1] Hence, by the binomial theorem,

$$J^n = (\sigma I + Z)^n = \sum_{k=0}^{K-1} \sigma^{n-k} \binom{n}{k} Z^k \qquad (n \geq K-1) \tag{11.5}$$

where we take $Z^0 = I$.[11.2] The case $\sigma = 0$ will be dealt with later; assuming $\sigma \neq 0$, we thus have

$$\begin{aligned} \lim_{n\to\infty} \sigma^{-n} \binom{n}{K-1}^{-1} J^n &= \lim_{n\to\infty} \sigma^{-n} \binom{n}{K-1}^{-1} (\sigma I + Z)^n \\ &= \sum_{k=0}^{K-1} \sigma^{-k} Z^k \lim_{n\to\infty} \binom{n}{K-1}^{-1} \binom{n}{k} = \sigma^{-K+1} Z^{K-1}; \end{aligned}$$

the last equation holds since the limit to the left of this equation is zero unless $k = K-1$. Now $Z^K \mathbf{e}_k = 0$ if $1 \leq k < K$ and

$$Z^{K-1}\mathbf{e}_k = \begin{cases} 0 & \text{if } 1 \leq k < K, \\ \mathbf{e}_1 & \text{if } k = K. \end{cases}$$

Hence, for any vector $\mathbf{x} = \sum_{k=1}^{K} \gamma_k \mathbf{e}_k$ we have

$$\lim_{n\to\infty} \sigma^{-n} \binom{n}{K-1}^{-1} J^n \mathbf{x} = \sigma^{-K+1} \gamma_K \mathbf{e}_1. \tag{11.6}$$

Noting that $J\mathbf{e}_1 = \sigma \mathbf{e}_1$, it follows that the vector on the right-hand side of this equation is an eigenvector of J unless $\gamma_K = 0$ (note that a Jordan block always has only one eigenvector aside from scalar multiples, and, of course, it has only one eigenvalue). If $\gamma_K \neq 0$, take $\mathbf{y} = \mathbf{x}$, and if $\gamma_K = 0$, take $\mathbf{y} = \mathbf{x} + \eta \mathbf{e}_K$ for an arbitrary $\eta \neq 0$. This establishes equations (11.1) and (11.2) with $\lambda = \sigma$ and

$$\alpha_n = \alpha_n(J) = \sigma^{-n} \binom{n}{K-1}^{-1} \qquad (\sigma \neq 0) \tag{11.7}$$

in case A is a single Jordan block of size $K \times K$ and eigenvalue $\sigma =\neq 0$.

For a single Jordan block with $\sigma = 0$ of size K, Lemma 11.1 does not directly apply, since we assumed that $\lambda \neq 0$. Nevertheless, the matrix A of the lemma may have a Jordan block associated with the eignevalue 0, so it is important to remark what happens in this case: equation (11.5) implies that $J^n = 0$ for $n \geq K$. □

To extend this proof to the general situation, it is better to talk in the language of linear transformations of a vector space into itself. To simplify our language, we will call a linear transformation of a vector space into itself a *linear operator*. To describe the proof for a general matrix, it is better to use the language of linear operators rather than that of a matrices. In the Jordan decomposition of a linear operator T on the vector space U, the space is written into a direct sum

$$U = \bigoplus_{l=1}^{M} U_l, \tag{11.8}$$

[11.1] That is, Z is a nilpotent matrix: a square matrix is called *nilpotent* if it has a power that is the zero matrix.
[11.2] For $n < K-1$ this formula is not valid, since then we should take n in that case as the upper limit of summation instead of $K-1$.

of independent subspaces U_l where each subspace is invariant for T, and on each U_l the restriction of T can be represented by a Jordan matrix as described in equation (11.3). For a detailed description, see the notes [3]: direct sums are described in [3, Definition 4.3, p. 9]. To describe this state of affairs is best to use the language of *projection operators* P_l. If for $\mathbf{u} \in U$ we have

$$\mathbf{u} = \sum_{l=1}^{M} \mathbf{u}_l \qquad (\mathbf{u}_l \in U_l \text{ for all } l \text{ with } 1 \le l \le n),$$

then we write $\mathbf{u}_l = P_l\mathbf{u}$. It is easy to see that $P_kP_lu = \delta_{kl}P_ku$ $(1 \le k, l \le M)$. The invariance for T of each U_l means that $T\mathbf{u}_l \in U_l$ for $u_l \in U_l$. In the language of projection operators, the invariance of each U_l can be expressed by saying that $TP_l\mathbf{u} = P_lTP_l\mathbf{u}$. Given that U_l is invariant for all l, we also have $P_lT\mathbf{u} = TP_l\mathbf{u}$ for all l and all $u \in U$; i.e., T and P_l commute.

11.1 Norms on vector spaces and the infinity norm

The problem with arguing with linear transformations rather than the matrices representing them is that such useful norms as the infinity norms are not available for abstract vector spaces. This turns out not to be much of the problem; the vector norms in different representations of the same abstract vector can be compared with the aid of inequality (10.4) and two different matrix representations of the same same linear tranformation by using the submultiplicative property (10.5). Indeed, if $\mathcal{X} = (x_1, x_2, ..., x_N)$ is one basis of the abstract vector space V and $\mathcal{Y} = (y_1, y_2, ..., y_N)$ is another basis, then there is a nonsingular $N \times N$ matrix S such that $\mathcal{X} = \mathcal{Y}S$,[11.3] and $\mathbf{x}$ and $\mathbf{y}$ are the representations of the same vector in the two different basis, then $\mathbf{y} = S\mathbf{x}$. Further, if A and B are matrix representations of in these bases of the same linear operator, then $A = S^{-1}BS$.[11.4] So using any norm on column vectors and the matrix norm induced by it, the norms of $\mathbf{x}$ and $\mathbf{y}$ are easy to compare:

$$\|\mathbf{y}\| = \|S\mathbf{x}\| \le \|S\|\|\mathbf{x}\| \qquad \text{and} \qquad \|\mathbf{x}\| = \|S^{-1}\mathbf{y}\| \le \|S^{-1}\|\|\mathbf{x}\|. \tag{11.9}$$

Similarly, for the matrices A and B:

$$\|A\| = \|S^{-1}BS\| \le \|S^{-1}\|\|B\|\|S\| \qquad \text{and} \qquad \|B\| = \|SAS^{-1}\| \le \|S\|\|A\|\|S^{-1}\|. \tag{11.10}$$

As a consequence of these inequalities, in many arguments it makes no difference which column vector or matrix representation of the abstract vectors or the abstract linear operators are used.

Proof of Lemma 11.1 for all matrices. Let $U = \mathbb{C}_{N,1}$ the space of N dimentional column vectors over the complex numbers $\mathbb{C}$, and let $T : U \to U$ be the linear operator represented by the basis consisting of the standard unit column vectors in this space.[11.5] Consider the decomposition of U to Jordan subspaces U_l $(1 \le l \le M)$ as in equation (11.8), and the eigenvalue σ_l be associated with the subspaces U_l for l with $1 \le l \le M$. Assume that for some L with $1 \le L \le M$ we have $\sigma_l = \lambda$ for l with $1 \le l \le L$ and $|\sigma_l| < \lambda$ for l with $L < l \le M$. Further, let K_l be the dimension of the space U_l, and assume that there is an L' with $1 \le L' \le L$ and a K such that $K_l = K$ for l with $1 \le l \le L'$ and $K_l < K$ for l with $L' < l \le L$. Finally, assume that each subspace U_l is spanned by the vectors

[11.3] The exact description of the matrix S is given in [3, Subsection 8.1 on p. 16]. The details and the meaning of the notation just used is not important for our present purposes; the only thing that is important that there is such a matrix S depending only on the two bases involved. The cited place uses P for the matrix denoted by S here.

[11.4] The matrix $S^{-1}BS$ is called a *similarity transformation* of the matrix B.

[11.5] A and T are essentially the same, most authors do not distiguish them. If one wants to distinguish them, A is a matrix while T is a mapping.

$\mathbf{e}_k^{(l)}$ with $1 \le k \le K_l$; here the vectors $\mathbf{e}_k$ used in the same sense as in the proof for Jordan blocks on p. 40 of Lemma 11.1. For an arbitrary $\mathbf{u} \in U$ write

$$\mathbf{u} = \sum_{l=1}^{M} \sum_{k=1}^{K_l} \gamma_k^{(l)} \mathbf{e}_k^{(l)},$$

where the $\gamma_k^{(l)}$ are appropriate complex scalars. According to equation (11.6) we have

$$\lim_{n\to\infty} \lambda^{-n} \binom{n}{K-1}^{-1} T^n \mathbf{u} = \sum_{l=1}^{L'} \lambda^{-K+1} \gamma_K^{(l)} \mathbf{e}_1^{(l)}.$$

The terms of the sum on the right-hand side for l with $L' < l < M$ were not needed, since

$$\lim_{n\to\infty} \frac{\sigma_l^n \binom{n}{K_l-1}}{\lambda^n \binom{n}{K-1}} = 0 \tag{11.11}$$

for l with $L' < l < M$.[11.6] Noting that the vectors $\mathbf{e}_1^l$ are independent, the right-hand side of (11.11) is not zero unless $\gamma_K^{(l)} = 0$ for all l with $1 \le l < L'$. The coefficient γ_K^1 can be made nonzero by adding a vector $\eta \mathbf{e}_K^{(1)}$ to $\mathbf{u}$. We will take

$$\alpha_n = \lambda^{i-n} \binom{n}{K-1}^{-1}; \tag{11.12}$$

with $\mathbf{u} = \mathbf{x}$ and $\mathbf{y} = \mathbf{u} + \eta \mathbf{e}_K^{(1)}$ for a small η (with $\eta = 0$ allowed)[11.7] in the notation given in the lemma. Equation (11.1) follows from this choice of α_n, and the existence of a nonzero limit in equation (11.2) also follows from our arguments. Equations (11.1) and (11.2) then imply that $A\mathbf{z} = \lambda \mathbf{z}$. If $\lambda > 0$, then equation (11.12) implies $\alpha_n > 0$, establishing last clause of the lemma, completing the proof. □

If we wanted to carry out the above proof in terms of the Jordan canonical form $B = SAS^{-1}$ of the matrix A, then in some calculations we would have to have used the matrix A^n and in others the matrix B^n. We have

$$B^n = \underbrace{(SAS^{-1})(SAS^{-1}) \dots (SAS^{-1})}_{n \text{ times}} = SA^nS^{-1}, \tag{11.13}$$

so the transition from A^n to B^n is simple.

12 The rest of the proof of the Perron–Frobenius theorem

Proof of Clause (b) of Theorem 10.1. We will use the power method described in Lemma 11.1 with vector $\mathbf{x}$ with positive entries; if necessary, we will change $\mathbf{x}$ slightly to the vector $\mathbf{y}$ so that the vector $\mathbf{y}$ will still have all positive entries. We will also make sure that positive $\alpha_n > 0$ for all

[11.6] While this formula is valid even in the special case $\sigma_l = 0$, the argument for that case is different: in that case T^n restricted to the subspace U_l is 0 for $n \ge K_l$ according to the comment made at the end of the proof of Lemma 11.1 for Jordan blocks.

[11.7] I.e., if we can take $\mathbf{y} = \mathbf{x}$.

$n \geq 0$.[12.1] Since A has all positive entries, $A^n\mathbf{y}$ will also have all positive entries, and so will have $\alpha_n A^n\mathbf{y}$, since $\alpha_n > 0$. The limit vector $\mathbf{z}$ obtained will have all nonnegative entries. Since all entries of A are positive, $A\mathbf{z}$ must have all positive entries; it cannot have a zero component. As $\mathbf{z}$ is an eigenvector for the eigenvalue r, we have $A\mathbf{z} = r\mathbf{z}$; hence $\mathbf{z}$ cannot have a zero entry. It is important to point out specifically that *A cannot have a eigenvector with nonnegative entries associated with a positive eigenvalue that has a zero entry*, □

Proof of Clause (c) of Theorem 10.1. Assume A has two eigenvectors $\mathbf{z}$ and $\mathbf{u}$ associated with the eigenvalue r, where $\mathbf{u}$ is not a scalar multiple of $\mathbf{z}$, and all entries of $\mathbf{z}$. Then there is a real α such that the vector $\mathbf{z} + \alpha\mathbf{u} \neq 0$ has all nonnegative entries but it also has at least one zero entry. Since this vector is also an eigenvector of A associated with the eigenvalue r, this is a contradiction, since we pointed out the the end of the of the proof of Clause (c) that a nonnegative eigenvector of A associated with the eigenvalue r cannot have a zero component. □

Proof of Clause (d) of Theorem 10.1. By replacing A with A/r, we may assume that the eigenvalue of the largest absolute value of A is 1. Let $\mathbf{z} = (z_1, z_2, \ldots, z_N)^T$ be an eigenvector having all positive associated with the eigenvalue 1 of A; that is, $A\mathbf{z} = \mathbf{z}$. Observe that for any $N \times N$ matrix $C = (c_{ij})$ with positive entries and any vector $\mathbf{y} = (y_1, y_2, \ldots, y_N)^T$ with positive entries we have

$$\begin{aligned}\|C\mathbf{y}\|_\infty &= \max\Big\{\sum_{j=1}^N c_{ij}y_j : 1 \leq i \leq N\Big\} \geq \max\Big\{\sum_{j=1}^N c_{ij} : 1 \leq i \leq N\Big\} \cdot \min\{y_j : 1 \leq j \leq N\} \\ &= \|C\|_\infty \cdot \min\{y_j : 1 \leq j \leq N\},\end{aligned}$$

where the last equality holds according to equation (10.6) saying that for any matrix, the infinity norm is the maximum of the row sum of the absolute values. Hence, noting that for any $n > 0$ all entries of A^n are positive and that all entries of $\mathbf{z}$ are positive, for all $n > 0$ we have

$$\|\mathbf{z}\|_\infty = \|A^n\mathbf{z}\|_\infty \geq \|A^n\|_\infty \cdot \min\{z_j : 1 \leq j \leq N\},$$

and so

$$\|A^n\|_\infty \leq \frac{\|\mathbf{z}\|_\infty}{\min\{z_j : 1 \leq j \leq N\}}. \tag{12.1}$$

This shows that $\|A^n\|$ remains bounded as $n \to \infty$.

On the other hand, if the largest eigenvalue 1 of A has a K-dimensional subspace for $K > 1$, as in equation (11.5) with $\sigma = 1$, for a Jordan block of K dimensions we have

$$J^n = (I + Z)^n = \sum_{k=0}^{K-1} \binom{n}{k} Z^k.$$

Here all the entries of Z^k for k with $1 \leq k \leq K$ are either 0 or 1, and so

$$\|J^n\|_\infty \geq \binom{n}{K-1} \geq n \qquad (K \geq 2). \tag{12.2}$$

[12.1] Since we are not thinking in terms of a practical calculation, it is not important how the choice of α_n is made. But, right after Lemma 11.1, we mentioned that α_n can be chosen as the first entry of $A^n\mathbf{y}$ among those having the largest absolute value; this choice will also ensure that $\alpha_n > 0$.

If J is a Jordan block of the matrix B, then

$$\|B^n\|_\infty \geq \|J^n\|_\infty \geq n \qquad (K \geq 2).$$

Now if B is the Jordan canonical form of A, then we have $B = SAS^{-1}$ for some invertible matrix S, and so $B^n = SAS^{-1}$ according to equation (11.13) and the discussion at the beginning of Subsection 11.1 Thus, assuming that B has a K-dimensional Jordan block associated with the eigenvalue $r = 1$, equations (12.1) and (12.2) contradict each other as $n \to \infty$; cf. (11.10) and (11.13). □

Proof of Clause (e) of Theorem 10.1. Applying Clause (b) of Theorem 10.1 to the matrix A^T, A^T has an column eigenvector $\mathbf{u}$ such that $A^T\mathbf{u} = \mathbf{u}$ with all positive entries; taking the transpose of this equation, we have $\mathbf{u}^T A = \mathbf{u}^T$. If $\mathbf{v}$ is an eigenvector of A associated with an eigenvalue $\sigma \neq 1$, then $A\mathbf{v} = \sigma\mathbf{v}$. Hence

$$\mathbf{u}^T\mathbf{v} = (\mathbf{u}A)\mathbf{v} = \mathbf{u}^T(A\mathbf{v}) = \mathbf{u}^T(\sigma\mathbf{v}) = \sigma\mathbf{u}^T\mathbf{v}.$$

Since $\mathbf{u}$ and $\mathbf{v}$ have all positive entries, $\mathbf{u}^T\mathbf{v}$ is a positive scalar; hence the above equation cannot hold for $\sigma \neq 1$. This shows that a positive eigenvector associated with an eigenvalue $\sigma \neq 1$ cannot exist. □

13 Differential versus difference equations

Differential and difference equations often behave very differently. We will illustrate this on the logistic differential and the logistic difference equations. As we mentioned was discussed above in the logistic difference equation above as equation (1.1). We modified that equation as

$$\Delta x_n = r(1 - x_n)x_n \qquad (x \geq 0), \tag{13.1}$$

where $\Delta x_n = x_{n+1} - x_n$, so we can compare it to a similar differential equation. By replacing x_n with

$$y_n = \frac{r+1}{r}x_n$$

this equation will become

$$y_n = (r+1)(1 - y_n)y_n,$$

which is identical to equation (1.1). We illustrate the behavior of equation (13.1) when $r = 2.9999$ and $x_0 = .2$. We ran the following maxima script; the script is almost identical to the script on p. 8 above, except for a slight change in the equation and except for some change in the names of variables and the plotting options:

```
kill(all);
linel : 60;
r : 2.9999;
xstart : .2;
x : xstart;
mylist : [];
loopto : 50;
for n : 0 step 1 thru loopto do (x_new : float(x+r*(1-x)*x),
  mylist : append(mylist,[[n,x]]), x : x_new);
n;
mylist : append(mylist,[[loopto+1,x_new]]);
```

```
12  plot2d([discrete,mylist],
13     [gnuplot_term,ps],
14     [style, [lines,5]],
15     [color, black],
16     [xlabel,"difference eq, r=2.9999, xstart=.2"],
17     [gnuplot_ps_term_command,
18      "set term eps size 5 in, 3 in"],
19     [gnuplot_out_file, "log_recr.eps"]);
```

We will not discuss the output of this script, since it is similar to earlier outputs discussed. The graph printed by the script can be found in Figure 13.1.

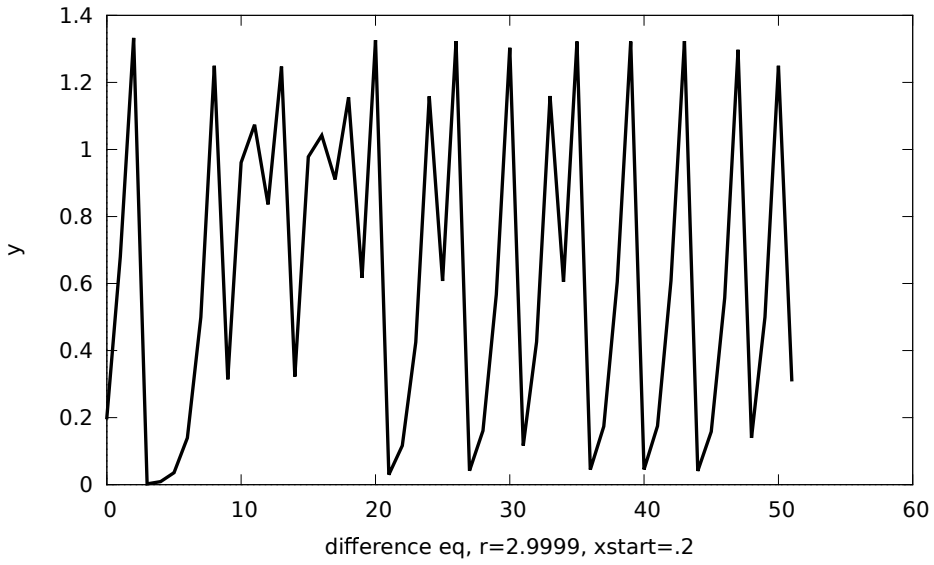

Figure 13.1: Logistic difference equation

The logistic differential equation is the equation

$$\frac{dx}{dt} = rx(1-x). \tag{13.2}$$

This equation is easily solved by analytical methods. The following script uses the classical Runge–Kutta method to solve this equation numerically for the same value $r = 2.9999$ and the same starting value $x = .2$ when $t = 0$, as above for the difference equation. This is accomplished by the following script:

```
1  kill(all);
2  linel : 60;
3  r : 3.2;
4  xstart : .2;
5  results : rk(r*x*(r-x),x,xstart,[t,0,10,0.1])$
6  plot2d([discrete,results],
7     [gnuplot_term,ps],
8     [style, [lines,5]],
```

```
 9     [color, black],
10     [xlabel,"differential eq, r=3.2, xstart=.2"],
11     [gnuplot_ps_term_command,
12      "set term eps size 5 in, 3 in"],
13     [gnuplot_out_file, "log_diff.eps"]);
```

The function `rk` on line 5 of the above script performs the Runge–Kutta method. The first argument `r*x*(r-x)` describes the function on the right-hand side of the differential equation (the left-hand side dx/dt is not needed), the second argument `x` identifies the dependent variable, the third arrument `xtart` gives the initial value, and the fourth argument is a list `[t,0,10,0.1]` giving the independent variable, its starting value, its final value, and the step size used in the method. The solution is displayed in Figure 13.2. While the difference equation displays wildly fluctuating, chaotic behavior, no such behavior is shown by the differential equation.

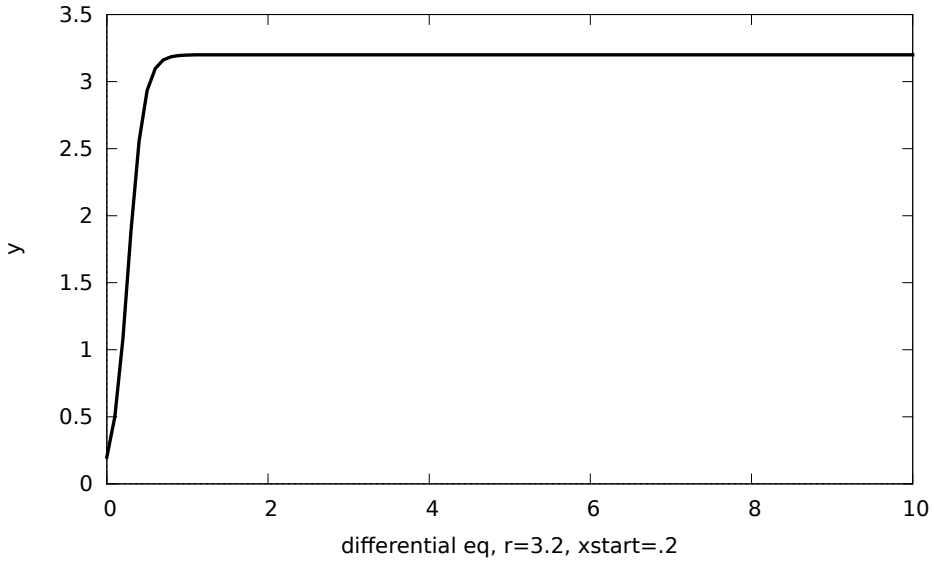

Figure 13.2: Logistic differential equation

14 Systems of autonomous first order linear differential equations with constant coefficients

A system of differential equation

$$y_k' = F_k(y_1, y_2, \ldots, y_n) \qquad (1 \le k \le n), \tag{14.1}$$

where each y_k is a function of a single independent variable x, and the prime $'$ indicates differentiation with respect to x is called *autonomous*, the term reflecting the fact that x does not directly occur in the equations. We will consider systems of autonomous first order linear differential equations with constant coefficients. Using D to denote the differential operator d/dx, in vector form these equations can be written as

$$D\mathbf{y} = A\mathbf{y}, \tag{14.2}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$ is a column vector of dependent variables and A is an $n \times n$ matrix with constant entries in $\mathbb{C}$. We are looking for functions $f_k(x)$ for k with $1 \le k \le n$ such that $y_k = f_k(x)$ satisfy this matrix equation. With a traditional abuse of notation, we will occasionally write $y_k(x)$ instead of $f_k(x)$, and we will also write $\mathbf{y}(x)$ if necessary. This problem can be handled by taking the Jordan canonical form B of the matrix A. The matrix B can obtained as a similarity transformation $B = SAS^{-1}$ of the matrix A, where S is an invertible $n \times n$ matrix. Writing $\mathbf{z} = S\mathbf{y}$, equation (14.2) can be written as $D\mathbf{z} = Bz$, which is especially easy to solve in view of the form of the matrix B. Then we can get the solution of the original equation (14.2) by noting that $\mathbf{y} = S^{-1}\mathbf{z}$.

The equation $\mathbf{z} = S\mathbf{y}$ can be written componentwise as

$$z_k = \sum_{j=1}^{n} s_{kj} y_j,$$

where z_k are entries of the column vector $\mathbf{z}$, and s_{kj} are entries of the matrix S; this means a linear change of variables. The matrix S needs to be invertible so that, after solving the transformed equations, we can return to the variables y_k. The Jordan canonical form of a matrix was discussion in Section 11; in particular, see equation (11.3) for the way a single Jordan block looks; see [3, Subsection 8.7] for details on the Jordan canonical form. $J = \sigma I + Z$ is a single Jordan block of size $K \times K$ as in equation equation (11.3), then we can use equations (11.4) to write the equation $D\mathbf{u} = J\mathbf{u}$ with $\mathbf{u} = (u_1, u_2, \ldots, u_K)$ Given that $\mathbf{u} = \sum_{k=1}^{n} u_k \mathbf{e}_k$, where $\mathbf{e}_k = (\delta_{k1}, \delta_{k2}, \ldots, \delta_{kn})^T$ is the kth unit vector, writing $u_{K+1} = 0$, we obtain

$$\begin{aligned}\sum_{k=1}^{K} Du_k \mathbf{e}_k = D\sum_{k=1}^{K} u_k \mathbf{e}_k = J\sum_{k=1}^{K} u_k \mathbf{e}_k \\ = \sum_{k=1}^{K} u_k J\mathbf{e}_k = \sum_{k=1}^{K} u_k(\sigma \mathbf{e}_k + \mathbf{e}_{k-1}) = \sum_{k=1}^{K} u_k \sigma \mathbf{e}_k + \sum_{k=1}^{K} u_k \mathbf{e}_{k-1}.\end{aligned}$$

Noting that $\mathbf{e}_0 = 0$ and $u_{K+1} = 0$, for the second sum on the right-hand side we have

$$\sum_{k=1}^{K} u_k \mathbf{e}_{k-1} = \sum_{k=2}^{K} u_k \mathbf{e}_{k-1} = \sum_{j=1}^{K-1} u_{j+1} \mathbf{e}_j = \sum_{j=1}^{K} u_{j+1} \mathbf{e}_j,$$

where the first equation holds because $\mathbf{e}_0 = 0$, the second one uses the substitution $j = k - 1$, and the third one holds because $u_{K+1} = 0$. Thus, the previous equation becomes

$$\sum_{k=1}^{K} (Du_k)\mathbf{e}_k = \sum_{k=1}^{K} (\sigma u_k + u_{k+1})\mathbf{e}_k;$$

the parentheses were added on the left for clarity – otherwise they do not make any difference, since differentiation does not affect the constant vector $\mathbf{e}_k$. The unit vectors $\mathbf{e}_k$ being linearly independent, the coefficients on the two sides of the last equation must be equal, resulting in the system of differential equations

$$u_k' = \sigma u_k + u_{k+1} \qquad (1 \le k \le K), \tag{14.3}$$

where $u_{K+1} = 0$. This system is easy to solve. Indeed, multiplying the equation by $e^{-\sigma x}$ and noting that for any function f of x we have

$$\big(f(x)e^{-\sigma x}\big)' = f'(x)e^{-\sigma x} - \sigma f(x)e^{-\sigma x},$$

according to the product rule of differentiation, the above system of equations can be rewritten as

$$\left(u_k e^{-\sigma x}\right)' = u_{k+1} e^{-\sigma x} \qquad (1 \le k \le K),$$

where $u_{K+1} = 0$. Hence, it is easy to see that the solution of the above system of equations is

$$u_k = e^{\sigma x} \frac{d^{k-1}}{dx^{k-1}} \sum_{j=0}^{K-1} c_j x^j \qquad (1 \le k \le K), \tag{14.4}$$

where the c_j for k with $0 \le j \le K-1$ are arbitrary constants.

14.1 Homogeneous linear differential equations with constant coefficients

Let y be a scalar variable of the independent variable x, and consider the differential equation

$$\sum_{k=0}^{n} a_k D^k y = 0 \qquad (a_n = 1). \tag{14.5}$$

Writing $y_1 = y$, this equation can be written as a system containing the dependent variables y_1, y_2, …, and y_n as

$$\begin{aligned} Dy_k &= y_{k+1} \qquad (1 \le k < n), \\ Dy_n &= -\sum_{k=1}^{n} a_{k-1} y_k. \end{aligned} \tag{14.6}$$

Writing $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$, this equation can be written in a matrix form as $D\mathbf{y} = A\mathbf{y}$, where

$$A = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & a_{n-2} & -a_{n-1} \end{pmatrix}.$$

This matrix is called the companion matrix of the polynomial in equation (14.5), that is, of the polynomial $P(t) = \sum_{k=0}^{n} a_k t^k$ of t, where $a_n = 1$; see [3, Subsection 8.4 on p. 18].[14.1] This approach allows one to discuss the solution of the equation (14.5) in terms of the solutions of systems of differenential equations. There are, however, several difficulties in this approach, which in the end makes the direct handling of equation (14.5) easier. The first problem is to find the Jordan canonical form of the matrix A. This problem is easily solved in that one can show that each zero of the polynomial $P(t)$ gives rise to a single Jordan block of the size the multiplicity of this zero. The second difficulty is to translate back the solution of the system of equations obtained in terms of the Jordan canonical form of A back to equation (14.5). Even this can be handled, but the amount of matrix theory needed for this makes the direct handling of equation (14.5) equation probably simpler. In any case, we do not need to discuss the solutions of this equation for our purposes.

[14.1] At the quoted location, the transpose of the matrix A given here is described as the companion matrix. Either version is common in the literature; in the present context the version listed here is more appropriate.

14.2 Matrix exponentiation

The system of differential equations with constant coefficients can also be solved by matrix exponentiation. The exponential of a square matrix A is derived as

$$\exp(A) = e^A \stackrel{def}{=} \sum_{m=0}^{\infty} \frac{1}{m!} A^m. \tag{14.7}$$

This is of course inspired by the series for the scalar exponential function

$$\exp(x) = e^x = \sum_{m=0}^{\infty} \frac{1}{m!} x^m.$$

Since the latter series converges for every value of x, one can show that the for that the former series converges for every square matrix A (e.g., by using matrix norms). If the matrices A and B commute, one can show that $\exp(A+B) = \exp(A)\exp(B)$. Indeed,

$$\begin{aligned} e^{A+B} &= \sum_{m=0}^{\infty} \frac{1}{m!}(A+B)^m = \sum_{m=0}^{\infty} \frac{1}{m!} \sum_{k=0}^{m} \binom{n}{k} A^k B^{m-k} = \sum_{m=0}^{\infty} \frac{1}{m!} \sum_{k=0}^{m} \frac{m!}{k!(m-k)!} A^k B^{m-k} \\ &= \sum_{m=0}^{\infty} \sum_{k=0}^{m} \frac{1}{k!} A^k \, \frac{1}{(m-k)!} B^{m-k} = \sum_{k=0}^{\infty} \frac{1}{k!} A^k \sum_{l=0}^{\infty} \frac{1}{l!} B^l = e^A e^B; \end{aligned} \tag{14.8}$$

the second equation here uses the binomial theorem, and the binomial theorem is not valid if A and A do not commute. If A and B do not commute, this equation may not be true; see [3, Subsection 9.6.1 on p. 28]. Similarly the way the scalar equation $y' = ay$ can be solved as $y = c\exp(ax)$, the vector equation $D\mathbf{y} = A\mathbf{y}$ can be solved as $\mathbf{y} = \exp(Ax)\mathbf{c}$, where x is the independent variable (a scalar, so we could have written xA instead of Ax), and $\mathbf{c}$ is a column vector. If the initial condition is given as $\mathbf{y} = \mathbf{c}$ for $x = 0$, then $\mathbf{y}(x) = \exp(Ax)\mathbf{c}$ for all x. This can be seen by differentiating both sides (the derivative of $\exp(Ax)$ can be obtained by differentiating the power series (14.7). This shows that we indeed found a solution of the equation $D\mathbf{y} = A\mathbf{y}$; that there is no other solution follows by standard uniqueness results for solutions of differential equations.

If $B = SAS^{-1}$ is the similarity transformation of the matrix A, then it immediately follows from equations (14.7) and (11.13) that $e^B = Se^AS^{-1}$. In case B is the Jordan canonical form of the matrix A, then one can get an easy understanding of the matrix exponential e^B, and it is easy to relate this exponential to the solution obtained in equation (14.4). In the special case when $J = \sigma I + Z$ is a Jordan block as in (11.3), then $e^J = e^\sigma e^Z$ according to equation (14.8). Since the matrix Z is nilpotent (see footnote 11.1 on p. 41), the series representing e^Z is finite, and it is easy to match up this series with the solution found in equation (14.4).

15 Equilibrium points of autonomous differential equations

Consider differential equation

$$\mathbf{y}' = \mathbf{F}(\mathbf{y}), \tag{15.1}$$

where $\mathbf{y} = (y_2, y_2, \ldots, y_n)^T$ is a column vector of dependent variables, with x being the independent variable and the prime indicating the differentiation d/dx, and $\mathbf{F}$ is a column vector of

functions: one can think of $\mathbf{F}$ as a column vector $(f_1, f_2, \ldots, f_n)^T$ where f_k is a multivariate function $f_k(y_1, y_2, \ldots, y_n)$.[15.1] A column vector $\mathbf{s}$ is called an *equilibrium point* of the above equation is $\mathbf{F}(\mathbf{s}) = 0$. In this case, $\mathbf{y}(x) = s$ for all x is a solution of the equation, called an *equilibrium solution*. If $\mathbf{F}$ is reasonably well behaved,[15.2] then this is the oly solution of the equation satisfying the initial condition $\mathbf{y}(x_0) = \mathbf{s}$ for some x_0. What we are interested in is what happens to a solution that satisfies this initial condition only approximately, but not exactly. The main question in this case is whether we will have $\lim_{x\to\infty} \mathbf{y}(x) = \mathbf{s}$; if so, then the equilibrium point $\mathbf{s}$ is called *stable*.

Many different types of behavior is possible, and it is difficult to obtain a general answer. All we will do there is to take a first order approximation

$$\mathbf{F}(\mathbf{s} + \mathbf{h}) \approx \mathbf{F}(\mathbf{s}) + A\mathbf{h} \tag{15.2}$$

for a small column vector $\mathbf{h}$, where A is an $n \times n$ matrix (incidentally, $\mathbf{F}(\mathbf{s}) = 0$ in the present case). More precisely, we seek a matrix A such that

$$\lim_{\mathbf{h}\to 0} \frac{\|\mathbf{F}(\mathbf{s} + \mathbf{h}) - \mathbf{F}(\mathbf{s}) - A\mathbf{h}\|}{\|\mathbf{h}\|} = 0.$$

The matrix A is called the *Jacobian matrix* of the function $\mathbf{F}$; see [6, Subsection 25.2 on p. 101] In a more general context, it is called the Fréchet derivative; see [6, Subsection 25.1 on p. 100].

We will study the main linear part

$$\mathbf{y}' = \mathbf{A}\mathbf{y} \tag{15.3}$$

of equation (15.1) near the point 0 to get an insight into the behavior of equation (15.1) near a point of equilibrium. The question as to how well the solutions of the two equations approximate each other will be left for a later discussion.

15.1 Two dimensional real valued cases

Assume A is a two-by-two matrix with real entries, and we will consider the behavior of equation (15.1) near an equilibrium point. We will assume that A is nonsingular; for singular matrices the approximation in equation(15.2) will not work, because then we will have $A\mathbf{h} = 0$ for some choices of $\mathbf{h} \neq 0$, and then higher order terms will dominate in the approximation. This means that 0 is not an eigenvalue of A. The behavior of the solution of equation near the point $\mathbf{y} = 0$ can be classified into several cases:

(a) Both eigenvalues of A are positive.

(b) Both eigenvalues of A are negative.

(c) A has one positive and one negative eigenvalue.

(d) Both eigenvalues of A are complex.

To simplify the notation, we will change the name of the independent variable to t, and write $\mathbf{y} = (x, y)^T$. The solution of equation (15.3) will be described as $x = x(t)$ and $y = y(t)$, and it can be visualized as a curve in the (x, y) plane, called the phase plane, as the parametric curve $\big(x(t), y(t)\big)$ for t in a certain range; we will be interested what happens when $t \to \infty$.

[15.1] Note that this is an abuse of language, since the latter is the value of a function at a certain place, the function itself being f_k.

[15.2] For example, if it is sufficient if the derivative of $\mathbf{F}$ is differentiable everywhere and its derivative is bounded.

15.1.1 The case of two positive eigenvalues

The Jordan canonical form B of the matrix A has form

$$B = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \tag{15.4}$$

unless $\lambda_1 = \lambda_2$. In case $\lambda_1 = \lambda_2$, this may be the form of the matrix B, or else B may be a Jordan block of size 2, when B will have the form

$$B = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}. \tag{15.5}$$

In the former case, the solution of the equation

$$(x', y') = B(x, y)^T \qquad \big(x(0) = x_0,\ y(0) = y_0\big) \tag{15.6}$$

is

$$\begin{aligned} x &= x_0 e^{\lambda_1 t}, \\ y &= y_0 e^{\lambda_2 t}. \end{aligned} \tag{15.7}$$

To describe the behavior near the equilibrium point $(0, 0)$, we need to choose x_0 and y_0 to be small positive or negative numbers. These equations are easy to graph. Assuming $t \geq 0$ (since we are interested what happens when $t \to \infty$), for the trajectory described by these equations we can write $u = e^{\lambda_1 t}$ and writing $\gamma = \lambda_2/\lambda_1$, these equations become

$$\left.\begin{aligned} x &= x_0 u \\ y &= y_0 u^{\gamma} \end{aligned}\right\} \qquad 1 \leq u < \infty. \tag{15.8}$$

Given that $\gamma > 0$, these equation describe a point moving away to infinity.

In case B is a Jordan block of size two as given in equation (15.5) the solution of equation (15.6) is

$$\begin{aligned} x &= (x_0 + y_0 t) e^{\lambda t}, \\ y &= y_0 e^{\lambda t}. \end{aligned} \tag{15.9}$$

One can try to use the substitution $u = e^{\lambda t}$ as before to visualize the trajectory described by these equations, though it is not as helpful here as before. In any case, the equations describe a curve moving away from the origin that deviates from the straight line only slightly.

15.1.2 The case of two negative eigenvalues

Equations (15.7) and (15.9) still apply in this case, but the eigenvalues being negative, the trajectories move toward the origin, i.e., the equilibrium point in this case on a curve. Instead of using the parametric representation in equation (15.8) is is better to write $u = e^{-\lambda_1 t}$ and $\gamma = \lambda_2/\lambda_1$ in this case, to obtain the parametric representation

$$\left.\begin{aligned} x &= x_0 u^{-1} \\ y &= y_0 u^{-\gamma} \end{aligned}\right\} \qquad 1 \leq u < \infty \tag{15.10}$$

in this case.

15.1.3 The case of a positive and a negative eigenvalue

Equation (15.7) still applies. The trajectories in this case move away from the origin as $t \to \infty$. If $\lambda_1 > 0$ and $\lambda_2 < 0$, the trajectories asymptotically move toward the x axis while moving away from the origin; it moves in the positive direction if $x_0 > 0$ and it moves in the negative direction if $x_0 < 0$. Having a size 2 Jordan block is not possible in this case, since for that the two eigenvalues must be equal. Equation (15.8) does apply in this case, except that $\gamma = \lambda_2/\lambda_1 < 0$ in this case.

15.1.4 The case of complex eigenvalues

Equation (15.7) still applies, but in this case the eigenvalues λ_1 and λ_2 are complex conjugates. This is because the matrix A has real entries, and so its characteristic polynomal real coefficients, and the eigenvalues are the zeros of this polynomial. The trouble in this case that in order to visualize the results, we need to give them in terms of real functions. The fact that earlier we moved from the original equation $\mathbf{y}' = A\mathbf{y}$ to a similar equation $\mathbf{y}' = B\mathbf{y}$ involving the Jordan canonical form B of the matrix A was not much of a problem, since such a transformation only involves a change of bases. The same linear transformation T is represented in one basis by the matrix A, in another basis by a matrix B. Whatever picture represents the solution involving the matrix B, the same picture, appropriately distorted by making the axes not perpendicular, represents the solution involving the matrix A. This is not true in case of the similarity transformation from A to B involves a matrix with complex entries. Therefore, we first need to translate the results obtained for the solution of the equation involving the matrix B to an equation involving a real matrix C that is similar to B.

Writing $\lambda_1 = \alpha + \beta i$ with real α and β (where $\beta \neq 0$, since λ_1 are not real), we must have $\lambda_2 = \alpha - \beta i$, and the Jordan canonical form of the matrix A has the form

$$B = \begin{pmatrix} \alpha + \beta i & 0 \\ 0 & \alpha - \beta i \end{pmatrix},$$

and the solution of equation (15.6) can be written as

$$\mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_0 e^{(\alpha+\beta i)t} \\ y_0 e^{(\alpha-\beta i)t} \end{pmatrix} \tag{15.11}$$

according to equation (15.7), where x_0 and y_0 are the initial values at $t = 0$. The simplest real matrix C that has the same eigenvalues of B is

$$C = \begin{pmatrix} \alpha & \beta \\ -\beta & \alpha \end{pmatrix}.$$

The matrix C is similar to the matrix B; indeed, we have $C = SBS^{-1}$, where

$$S = \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \qquad \text{and} \qquad S^{-1} = \frac{1}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}.$$

Writing $\mathbf{u} = S\mathbf{y}$, it is easy to see that $\mathbf{y}$ is a solution of the equation $\mathbf{y}' = B\mathbf{y}$ if and only if $\mathbf{u}$ is a solution of the equation $\mathbf{u}' = C\mathbf{u}$. With the solution for $\mathbf{y}$ given in equation (15.11), this gives

$$\begin{aligned} \mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix} &= \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}\begin{pmatrix} x_0 e^{(\alpha+\beta i)t} \\ y_0 e^{(\alpha-\beta i)t} \end{pmatrix} \\ &= \begin{pmatrix} x_0 e^{(\alpha+\beta i)t} + i y_0 e^{(\alpha-\beta i)t} \\ i x_0 e^{(\alpha+\beta i)t} + y_0 e^{(\alpha-\beta i)t} \end{pmatrix}. \end{aligned} \tag{15.12}$$

A necessary and sufficient condition for the vector on the right-hand side be real is that $iy_0 = \overline{x_0}$, where the bar indicates complex conjugate. Indeed, for the two complex numbers $\alpha = x_0 e^{(\alpha+\beta i)t}$ and $\beta = iy_0 e^{(\alpha-\beta i)t}$, the expressions $\alpha+\beta = x_0 e^{(\alpha+\beta i)t} + iy_0 e^{(\alpha-\beta i)t}$ and $i(\alpha-\beta) = ix_0 e^{(\alpha+\beta i)t} + y_0 e^{(\alpha-\beta i)t}$ are both real if and only if α and β are conjugate to each other. According to Euler's equation

$$e^{it} = \cos t + i \sin t, \tag{15.13}$$

the numbers $e^{(\alpha+\beta i)t}$ and $e^{(\alpha-\beta i)t}$ are conjugate to each other (for Euler's equation, see [5, equation (15) on p. 6]); so, indeed, x_0 and iy_0 must also be conjugate to each other.

Assuming $iy_0 = \overline{x_0}$, to express the solution given in equation (15.12) in terms of trigonometric functions, assume $x_0 = u_0 e^{i\eta}$ for some real u_0 and η, we have $y_0 = -iu_0 e^{-i\eta}$. Noting that for any complex number z we have $z + \bar{z} = 2\Re z$, we obtain

$$\mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_0 e^{(\alpha+\beta i)t} + iy_0 e^{(\alpha-\beta i)t} \\ ix_0 e^{(\alpha+\beta i)t} + y_0 e^{(\alpha-\beta i)t} \end{pmatrix} = \begin{pmatrix} 2u_0 e^{\alpha t} \cos(\beta+\eta)t \\ -2u_0 e^{\alpha t} \sin(\beta+\eta)t \end{pmatrix}.$$

For $\alpha = 0$, these equations describe a circle in the (u, v)-plane; for increasing t the circle is traversed clockwise.[15.3] For $\alpha \neq 0$ it is a logarithmic spiral. For increasing t, the point on the curve is moving away from the origin to infinity if $\alpha > 0$, and it moves toward the origin if $\alpha < 0$.

15.2 Comparing the linear equation to the original equation

The Hartman-Grobman theorem asserts that if the matrix A has no eigenvalue whose real part is 0, then, assuming that 0 is an equilibrium point, the qualitative behavior near the equilibrium point 0 of the linearized equation $\mathbf{y}' = A\mathbf{y}$ is the same as the qualitative behavior of the equation $\mathbf{y}' = \mathbf{F}(\mathbf{y})$, assuming that $\mathbf{F}$ is a smooth function close to the equilibrium point. Here smoothness means repeated differentiability; we will not formulate this result precisely, and its proof is beyond the scope of these notes. One note of caution: the sameness of qualitative behavior does not mean that the solution of the linearized equation is close to the solution of the original equation.

At the website of Alun Loyd of North Carolina State University, there are interesting pictures illustrating the different behaviors of a system of two differential equations.

15.3 Finding the Jordan canonical form of a matrix

In order to find the Jordan canonical form of a matrix, one first needs to find its minimal polynomial, defined in [4, Section 4 on p. 7], where a method for finding the minimal polynomial is also described in Subsection 4.4 on p. 9. A worked out example showing how to find the minimal polynomial is given in Section 6 on p. 13. This example is continued in Section 7 on p. 14 to find the Jordan canonical form of the matrix.

[15.3] The case $\alpha = 0$ is exceptional, since in this case the qualitative behavior of the linear equation may not reflect the behavior of the original equation. See Subsection 15.2 next.

16 Stochastic processes

16.1 Random walk

Let X_n for $n > 0$ be independent random variables, and assume each X_n is $+1$ or -1, assuming each of these values with probability $1/2$. Let

$$Y_n = \sum_{k=1}^{n} X_k \qquad (n \geq 0)$$

(for $n = 0$ this sum is the empty sum, defined to be 0). The sequence $\{Y_n\}$ of random variables is called a simple one-dimensional *random walk*. One can interpret this as going for a walk along the number time, starting at the point 0 at time $t = 0$, and at each integer time deciding to move right or left with probability $1/2$. A generalization to higher dimensions is straightforward. For example, for a two-diminsional random walk, one starts with independent pairs of random variables (U_k, V_k) for all $k > 0$ such that given k, each of the events $(U_k, V_k) = (1, 0)$, $(U_k, V_k) = (-1, 0)$, $(U_k, V_k) = (0, 1)$, and $(U_k, V_k) = (0, -1)$ has probability $1/4$.[16.1]

$$X_n = \sum_{k=1}^{n} U_k,$$
$$Y_n = \sum_{k=1}^{n} V_k$$

for $n \geq 0$.

16.2 Wiener process

Definition 16.1. A Wiener process, named after Norbert Wiener, is a sequence of random variables W_t for $t \geq 0$ such that

(a) $W_0 = 0$ almost surely.[16.2]

(b) The process has independent increments; i.e., for any t_1, t_2, ..., t_n for $n > 2$, with $t_k < t_1 < t_2$ for any k with $2 < k \leq n$ the variables $W_{t_2} - W_{t_1}$ is independent of the sequence of variables $\langle W_k : 2 < k \leq n \rangle$.

(c) For any $t > 0$ and $\delta > 0$, the increment $W_{t+\delta} - W_t$ is normally distributed with mean 0 and variance δ.

(d) W_t is almost surely a continuous function of t.

A Wiener process is also called a one-dimensional *Brownian motion*; the name is inspired by the natural motion of particles suspended in a fluid, a phenomenon observed in nature and originally called Brownian motion. The motion of the particles is due to their collision with the molecules of the fluid. There are straightforward natural generalizations to higher dimensions of the one-dimensional Brownian motion.

[16.1] Of course, U_k and V_k are not independent of each other, according to this description. What independence means here is that the probability of arbitrarily chosen outcomes of these pairs for m given values of k is 4^{-m}.
[16.2] I.e., with with probability 1; *almost surely* is usually abbreviated as a.s.

If Y_n is a random walk, and

$$W_n(t) = \frac{1}{\sqrt{n}} Y_{\lfloor nt \rfloor} \qquad \text{for} \quad t \in [0,1] \quad \text{and} \quad n \geq 0,$$

then W_n approaches a Wiener process as $n \to \infty$. This is a special case of Donsker's theorem.

16.3 Integration

Given two sequences of $H(t)$ and $W(t)$ of random variables, one can try to define a Riemann–Stieltjes type integral as follows.[16.3] To define such an integral, we first need the definition of a partition:

Definition 16.2 (Partition)**.** A *partition* of the interval $[a, b]$ is a finite sequence $\langle t_0, t_1, \ldots, t_n \rangle$ of points such that

$$P : a = t_0 < t_1 < t_2 < \ldots < t_n = b.$$

The *width* or *norm* of a partition is

$$\|P\| \stackrel{def}{=} \max\{t_i - t_{i-1} : 1 \leq i \leq n\}.$$

Assuming $0 < a < b$, associated with such a partition P is a *Riemann–Stieltjes* sum:[16.4]

$$S_P = \sum_{i=1}^{n} H(t_{i-1})\big(W(t_i) - W(t_{i-1})\big). \tag{16.1}$$

One takes the limit of such sums when $\|P\| \to 0$; the limit can be taken in probability. This means that there is a random variable Z such that for every $\epsilon > 0$ there is a $\delta > 0$ such that $\mathrm{P}(|S_P - Z| < \epsilon) > 1 - \epsilon$ if $\|P\| < \delta$. If this limit Z exists in the sense described, then the random variable Z is called the *Itô integral* of H with respect to W on the interval $[a, b]$, and written as

$$\int_a^b H(t)\, dW(t). \tag{16.2}$$

For more on stochastic integration, see e.g. [8].

16.4 The Kolmogorov probability model

In 1933, Kolmogorov introduced the modern foundation of probability theory. According to this, a probability space is an ordered triple $\langle S, \mathcal{B}, \mathrm{P} \rangle$, where the nonempty set S is the underlying spacem $\mathcal{B}$ is a σ-algebra of subsets of S, and $\mathrm{P} : \mathcal{B} \to [0, 1]$ is a probability measure. No assumptions are made about S. $\mathcal{B}$ has the following properties:

Definition 16.3. . Given a set S, a set $\mathcal{B}$ of subsets of S is called a σ-algebra on S if the following conditions are satisfied.

(a) We have $\emptyset \in \mathcal{B}$.

(b) If $A \in \mathcal{B}$ then $S \setminus A \in \mathcal{B}$.

[16.3] The Riemann–Stieltjes integral is a generalization by Stieltjes of an earlier integral concept by Riemann.

[16.4] Note that in the sum to follow we take the value of H at the left endpoint of the interval $[t_{i-1}, t_i]$, whereas in the Riemann–Stieltjes sum for ordinary functions, the function value is taken at an arbitrarily selected point $\xi_i \in [t_{i-1}, t_i]$, called the tag associated with this interal. There is an important reason for this, to be explained below.

(c) If $A_n \in \mathcal{B}$ for all integers $n > 0$, then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{B}$.

The sets in $\mathcal{B}$ are called events, i.e., sets for which probability is defined. Note that these axioms imply that if $A, B \in \mathcal{B}$ then $A \cup B \in \mathcal{B}$ according to Clause (c) by taking $A_1 = A$, $A_2 = B$, and $A_n = \emptyset$ for $n \geq 3$. Further, we also have $A \cap B \in \mathcal{B}$ under the same assumption. Indeed, we have

$$A \cap B = S \setminus \big((S \setminus A) \cup (S \setminus B)\big).$$

As for the probability measure P, we stipulate

Definition 16.4. Given a σ-algebra $\mathcal{B}$ on the set S, the function $\mathrm{P} :\to [0, 1]$ is called a probability measure if

(a) We have $\mathrm{P}(S) = 1$.

(b) We have $\mathrm{P}(\emptyset) = 0$.

(c) If $A_n \in \mathcal{B}$ for all integers $n > 0$ and $A_m \cap A_n = \emptyset$ whenever $0 < m < n$, then

$$\mathrm{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathrm{P}(A_n).$$

Note that Clause (b) is unnecessary; for example, it follows from Clause (c) by taking $A_n = \emptyset$ for all $n > 0$ and noting that $\mathrm{P}(\emptyset)$ is a real number.

A (real-valued) *random variable* on the probability space $\langle S, \mathcal{B}, \mathrm{P}\rangle$ is a function $X : S \to \mathbb{R}$ such that for all $x \in R$ we have $\{s \in S : X(s) \leq r\} \in \mathcal{B}$ (so it is meaningful to talk about the probability of this set. As a consequence, we also have $\{s \in S : X(s) < r\} \in \mathcal{B}$, $\{s \in S : X(s) = r\} \in \mathcal{B}$, and $\{s \in S : X(s) > r\} \in \mathcal{B}$. If one develops integration theory, one can define the expectation of the random variable X as

$$\mathrm{E}(X) = \int_S X(s)\, d\,\mathrm{P}(s);$$

we will not explain the meaning of this, merely mention that this approach enables a more elegant discussion of expectation than the the usual approach in undergraduate courses given in order to avoid the need to develop integration theory.[16.5] This latter approach introduces the distribution function

$$F_X(x) = \mathrm{P}(\{s \in S : X(x) \leq r\}),$$

and then defines the expectation as a Stieltjes integral

$$\mathrm{E}(X) = \int_{-\infty}^{\infty} x\, dF_X(x).$$

Actually, even this is not done; instead one discusses discrete and continuos random varialbles, and then expectation can be defined in terms of a usual improper integral of the density function in case of a continous random variable, and as a sum in case of a discrete random variable.

It is interesting to note that when Lebesgue developed his integration theory, he first defined the measure[16.6] of sets belonging to a certain σ-algebra $\mathcal{L}$ of real numbers, and then considered only functions f, called measurable, for which the set $\{x : f(x) \leq \lambda\}$ is measurable, i.e., belongs to $\mathcal{L}$.

[16.5] Of course, this integral does not always exist, in which case the random variable X does not have an expectation.

[16.6] The measure of an interval is its length. The point is to develop a theory that gives the measure of more complicated sets forming a σ-algebra containg all intervals.

Given a finite interval, letting $m(\lambda)$ be the measure of the set $\{x \in [a,b] : f(x) \leq \lambda\}$,[16.7] he defined the integral of f on $[a,b]$ as the Stieltjes integral

$$\int_{-\infty}^{\infty} \lambda \, dm(\lambda),$$

assuming this Stieltjes integral is absolutely convergent. Note that this is exactly the same as the way used to calculate the expectation of a random variable, given its distribution function.[16.8] However, since them more elegant definitions have been found.

16.5 Filtration

Definition 16.5 (Filtration)**.** Given a probability space $\langle S, \mathcal{B}, \mathrm{P} \rangle$, a filtration is a sequence of σ-algebras $\{\mathcal{B}_t\}_{t \geq 0}$ such that $\mathcal{B}_{t_1} \subset \mathcal{B}_{t_2} \subset \mathcal{B}$ whenever $0 \leq t_1 \leq t_2$.

Definition 16.6 (Adapted process)**.** The random process $\{X_t\}_{t \geq 0}$ is adapted to the filtration if for each $t \geq 0$, the random variable X_t is measurable for $\mathcal{B}_t$; that is, for every $x \in \mathbb{R}$ we have $\{s \in S : X_t(s) \leq s\} \in \mathcal{B}_t$.

While we are not going into the rather subtle mathematical theory involving filtrations and Itô integrations, we wanted to mention them since their intuitive meaning can be expained in terms of financial markets: When one considers the price of a stock as a random variable, it is reasonable to assume that this price does not depend on information that will only be available in the future. This property of a stochastic process is expressed by the concept of *filtration*. The σ-algebra $\mathcal{B}_t$ is meant to describe all the factors (or information) available at the present that can influence the price of a stock; for example, an unpredictable earthquake in the future cannot influence stock prices at present. So the random variable X_t modeling the stock price must be measurable in terms of $\mathcal{B}_t$. On the other hand, if $0 < t_1 < t_2$, then X_{t_2}, describing price of the stock at time t_2 need not be measurable in terms of $\mathcal{B}_{t_1}$, i.e., the information available at time t_1.[16.9]

Definition 16.7 (Natural filtration)**.** Given a stochastic process $\{X_t\}_{t \geq 0}$, the *natural filtration* is the sequence $\{\mathcal{B}_t\}$ of σ-algebras, where $\mathcal{B}_t$ for $t \geq 0$ is the σ-algebra generated by all the sets $\{s \in S : X_{t'}(s) \leq x\}$ for all $t' \leq t$ and $x \in \mathbb{R}$. This is also called the filtration generatied by the stochastic process $\{X_t\}_{t \geq 0}$.

The natural filtration of a process in a way characterizes the past behavior of the process.

16.6 Comment on the Ito integral

In discussing the Ito integral described in formula (16.2) one usually assumes that the process $H(t)$ is adapted to the natural filtration of the process $W(t)$. In the light of this, we can now make sense of the requirement that in the sum in equation (16.1) one takes the value of H at the beginning of the interval. This means that one first measures the value of H at time t_{i-1} when the unpredictable random change $W_{t_i} - W_{t_{i-1}}$ is still in the future.

[16.7] We need to take a finite interval to ensure that $m(\lambda)$ is finite.

[16.8] Even the requirement of absolute convergence is present also for expectation.

[16.9] This point is somewhat subtle, since one can make predictions for X_{t_2} at time t_1. Such a prediction can be expressed in terms of another random variable that is measurable in terms of $\mathbb{B}_{t_1}$.

17 Stochastic differential equations

We will consider differential equations of form

$$dX_t = f(X_t, t)\,dt + g(X_t, t)\,dW_t, \tag{17.1}$$

where X_t is a stochastic process, W_t is a Wiener process, and f and g are real-valued functions of two real variables. This differential equation is interpreted in terms of its integral form:

$$X_{t_2} - X_{t_1} = \int_{t_1}^{t_2} f(X_t, t)\,dt + \int_{t_1}^{t_2} g(X_t, t)\,dW_t \qquad (t_1 < t_2),$$

where the second integral on the right-hand side is taken as an Itô integral.

17.1 Itô's lemma

Lemma 17.1 (Itô)**.** *Assume the stochastic process X_t satisfies equation (17.1), where W_t is a Wiener process. Let $F(x,t)$ be a real-valued function of two real variables, and assume that F is twice continuously differentiable.*[17.1] *Then*

$$dF(X_t, t) = \left(F_t + F_x f + \frac{g^2}{2} F_{xx}\right) dt + g F_x\,dW_t. \tag{17.2}$$

On the right-hand side, the arguments of the partial derivatives of F, and of the functions f and g are X_t and t; we suppressed these arguments for the sake of clarity.

Outline of proof. Using Taylor's formula of two variables (see e.g. [2, Section 28, p. 107], we have

$$\begin{aligned} F(X_t + dX_t, t + dt) - F(X_t, t) &= F_x\,dX_t + F_t\,dt \\ &\quad + \frac{1}{2}(F_{xx}\,dX_t^2 + 2F_x F_t\,dX_t\,dt + F_{tt}\,dt^2) + \dots \\ &= F_x\,dX_t + F_t\,dt + \frac{1}{2}\,F_{xx}\,dX_t^2 + \dots, \end{aligned}$$

where the terms on the right that tend to 0 when divided by dt were omitted. Substituting dX_t from equation (17.1) on the right-hand side, we obtain

$$\begin{aligned} F(X_t + dX_t, t + dt) - F(X_t) &= F_x f\,dt + F_x g\,dW_t + F_t\,dt \\ &\quad + \frac{F_{xx}}{2}(f^2\,dt^2 + 2fg\,dt\,dW_t + g^2\,dW_t^2) + \dots \\ &= F_x f\,dt + F_x g\,dW_t + F_t\,dt + \frac{1}{2}F_{xx}g^2\,dW_t^2 + \dots \end{aligned} \tag{17.3}$$

On the right-hand side, we substitute dW_t^2 with dt to obtain equation (17.2). The justification for this substitution is is somewhat subtle, and it is based on Clause (c) in Definition 16.1, the expectation of dW_t^2 is dt. We will outline an explanation. Given T_1 and T_2 with $0 \le a < b$,[17.2] we take a partition

$$P : a = t_0 < t_1 < t_2 < \dots < t_n = b,$$

[17.1] The function is really F and not $F(x,t)$, but we need to indicate the variables to use a customary notation for its partial derivatives.

[17.2] The need for the assumption $a \ge 0$ is that we assumed that the process W starts at time 0.

then

$$\int_a^b (dW_t)^2 \stackrel{def}{=} \lim_{\|P\|\to 0} \sum_{i=1}^n \big(W(t_i) - W(t_{i-1})\big)^2 = \sum_{i=1}^n (t_i - t_{i-1}) = b - a,$$

as we will explain. We wrote $\stackrel{def}{=}$ after the integral, since the integral on the left was not defined as an Itô integral. The limit we can mean in the sense of probability, as in the definition of the Itô integral. The expectation of the ith term in the sum is the variance of $W(t_i) - W(t_{i-1})$, which is $t_i - t_{i-1}$ according to Clause (c) in Definition 16.1. The terms of this sum are independent random variables according to Clause (b) of the same definition. Adding them up, the sum will converge to the sum of expectation, as a kind of law of large numbers.

To explain this in more detail, we have

$$\mathrm{E}\Big(\sum_{i=1}^n \big(W(t_i) - W(t_{i-1})\big)^2\Big) = \sum_{i=1}^n \mathrm{E}\left(\big(W(t_i) - W(t_{i-1})\big)^2\right) = \sum_{i=1}^n (t_i - t_{i-1}) = b - a,$$

Further, as we mentioned, $W(t_i) - W(t_{i-1})$ is an $\mathcal{N}(0, t_i - t_{i-1})$ variable, i.e., if Z is a standard normal variable, then the distribution of $W(t_i) - W(t_{i-1})$ is that of the distribution of $\sqrt{t_i - t_{i-1}}\, Z$, and so the distribution of $\big(W(t_i) - W(t_{i-1})\big)^2$ is that of the distribution $(t_i) - t_{i-1})Z^2$. The distribution of Z^2 is a χ_1^2 variable; this has variance 2. Hence $\big(W(t_i) - W(t_{i-1})\big)^2$ has variance $2(t_i - t_{i-1})^2$. Thus

$$\begin{aligned}\mathrm{Var}\Big(\sum_{i=1}^n \big(W(t_i) - W(t_{i-1})\big)^2\Big) &= \sum_{i=1}^n \mathrm{Var}\left(\big(W(t_i) - W(t_{i-1})\big)^2\right) \\ &= 2\sum_{i=1}^n (t_i - t_{i-1})^2 \le 2\sum_{i=1}^n \|P\|(t_i - t_{i-1}) = 2\|P\|(b-a),\end{aligned}$$

which tends to 0 as $\|P\| \to 0$. Here the first equation follows since the variance of a sum of independent variables equals the sum of the variances. the inequality follows since $t_i - t_{i-1} \le \|P\|$, and so

$$(t_i - t_{i-1})^2 = (t_i - t_{i-1})(t_i - t_{i-1}) \le \|P\|(t_i - t_{i-1}).$$

Since the variance of the sum

$$\sum_{i=1}^n \big(W(t_i) - W(t_{i-1})\big)^2$$

tends to 0, the sum itself tends to its expectation in probability.

As a consequence, one can show that

$$\int_a^b g(X_t, t) F_x(X_t, t)\, dW_t^2 = \int_a^b g(X_t, t) F_x(X_t, t)\, dt,$$

where the integral on the left is to be interpreted as an Itô integral, the intergral on the right is just an ordinary integral (with a random process as the integrand). Given that equation (17.1) is interpreted as an Itô integral justifiles replacing dW_t^2 with dt in equation (17.3).[17.3] □

[17.3] See [12] for more details.

17.2 The Stratonovich integral

If we modify the approximating sum (16.1) of the ito integral to

$$S_P = \sum_{i=1}^{n} \frac{H(t_{i-1}) + H(t_i)}{2} \big(W(t_i) - W(t_{i-1})\big),$$

if these sums converge, the random variable that they converge to is called the Stratonovich integral

$$\int_a^b H(t) \circ dW(t).$$

The stochastic differential equation analogous to equation (17.1) is written as

$$dX_t = f(X_t, t)\, dt + g(X_t, t) \circ dW_t;$$

note the symbol $\circ$ preceding with dW_t is both the integral and the differential equation. This differential equation is to be interpreted with the Stratonovich integral.

There are important differences between the Itô and the Stratonovich integral; there are formulas that express one in terms of the other. The Itô integral is preferred in the financial industry and in numerical approximations, and the Stratonivich integral is often preferred in physics. Normal calculus rules can be used with the Stratonovich integral; many of the standard integration techniques also apply. Not so with the Itô integral, as shown by Itô's Lemma 17.1.

Given that the Stratonovich integral is easier to manipulate since it follows normal calculus rules, one may ask why one does not always prefer it to the Itô integral, at least in theoretical calculations. The answer to this questions is that the possibility of translation between the Stratonovich integral and the Itô integral relies strictly on the assumptions that makes the process $W(t)$ a Wiener process.[17.4] When one discusses the motion of small particles in air bombarded by air molecules, these conditions are more accurately satisfied than for stock prices moving under various influences; therefore, the Stratonivich integral is appropriate only for the former, because the random nature of the process $W(t)$ at the beginning of the time interval $[t_{i-1}, t_i]$ is not much different from its behavior at the end. This assumption cannot be made for stock prices.

18 The Euler–Maruyama method

The form of the approximating sums for the Itô integral in equation (16.1) directly lends itself to the numerical approximation of the Itô integral by simulating the random proces W, and along with it, to an approximation of the solution of a stochastic differential equation (17.1) given in the Itô form. Taking this equation on the interval $[0, T]$, with initial condition $X_0 = x_0$, we divide the interval $[0, T]$ into n equal parts with $t_k = kT/n$ for k with $0 \le k \le n$, and taking ΔW_k to be a random number with normal distribution $\mathcal{N}(0, 1/n)$, that is, with expectation 0 and variance $1/n$, we put

$$x_{t_k} = x_{t_{k-1}} + f(x_{t_{k-1}}, t_{k-1})\, \frac{1}{n} + g(x_{t_{k-1}}, t_{k-1}) \Delta W_k \qquad (1 \le k \le n).$$

[17.4] These integrals are applicable to a wider class of processes than just Wiener processes, but we want to avoid technicalities. Further, assumptions need also to be made about the integrand $H(t)$ for the integral to exist.

18.1 Geometric Brownian motion

The solution of the differential equation

$$dX_t = \mu X_t\,dt + \sigma X_t\,dW_t, \tag{18.1}$$

where μ and $\sigma > 0$ are constants, is called a one-dimensional *geometric Brownian motion.* [18.1] Here μ is called the *percentage drift*, and σ, the percentage volatility.[18.2]

18.2 A Maxima implementation of the Euler–Maruyama method

The following program implements the Euler–Maruyama method to solve the equation (18.1) with $\mu = 0.1$, $\sigma = 0.15$, and presents the graphs of two runs of the solution on the interval $[0, 10]$ using 150 steps.

```
kill(all);
linel : 60;
load(distrib);
mu : 0.1;
sigma : 0.15;
tlim : 10;
tstart : 0;
xstart : 1;
loopto : 150;
deltat : tlim/loopto;
st : make_random_state(56088371546)$
set_random_state(st);
for i : 1 step 1 thru 2 do(
t : tstart,
x : xstart,
mylist : [],
for k : 0 step 1 thru loopto
do (dw : random_normal(0,sqrt(deltat)),
  xnew : float(x+mu*x*deltat+sigma*x*dw),
  tnew : float(t+deltat),
  mylist : append(mylist,[[t,x]]),
  x : xnew, t : tnew),
mylist : append(mylist,[[t,x]]),
mylists[i] : mylist);
plot2d(
   [[discrete,mylists[1]],
   [discrete,mylists[2]]],
   [gnuplot_term,ps],
   [style, [lines,5]],
   [color, red, blue],
   [xlabel,"t"],
   [ylabel,"x"],
```

[18.1] If we take $\sigma = 0$, we are not dealing with a random process. We could allow $\sigma < 0$, but that does not give rise to a qualitatively different stochastic process, since the distribution of dW_t is a normal distribution with expectation 0, and so it is symmetric about the origin.

[18.2] The word "percentage" here is a nonsense term unless these quantities are given as as percentages. Better terms would be "drift factor" and "volatility factor," or "multiplicative drift" and "multiplicative volatility," or simply "drift" and "volatility."

```
33      [gnuplot_ps_term_command,
34       "set term eps size 5 in, 3 in"],
35      [gnuplot_out_file, "euler_mar.eps"]);
```

The numbers at the beginnings of the lines are line numbers as usual, and they are not part of the program. Very few new Maxima features are included in the program, so only a short description will suffice. Line 3 loads the package `distrib`. On Line 11 a new random state is created and on line 12 this state is assigned to the random generator. As mentioned before, this is important when testing the program, so any two runs of the program produces the same result. The outside loop starting on line 13 and ending on line 24 gives two runs of the solution of the equation for geometric Brownian motion described in the loop starting on lines 18–22. The command on line 18 creates a random variable with mean 0 and standard deviation `sqrt(deltat)`, corresponding to $\sqrt{dt}$; i.e., of variance dt, which is what it should be for a Wiener process, according to Clause (c) in Definition 16.1. The datapoints of these two runs are stored in the list `mylists` with two members `mylists[1]` and `mylists[2]`. In lines 25–35, the two graphs are printed out, the first one in red and the second one in blue; the color specifications are given on line 30. The printed out figure is in Figure 18.1.

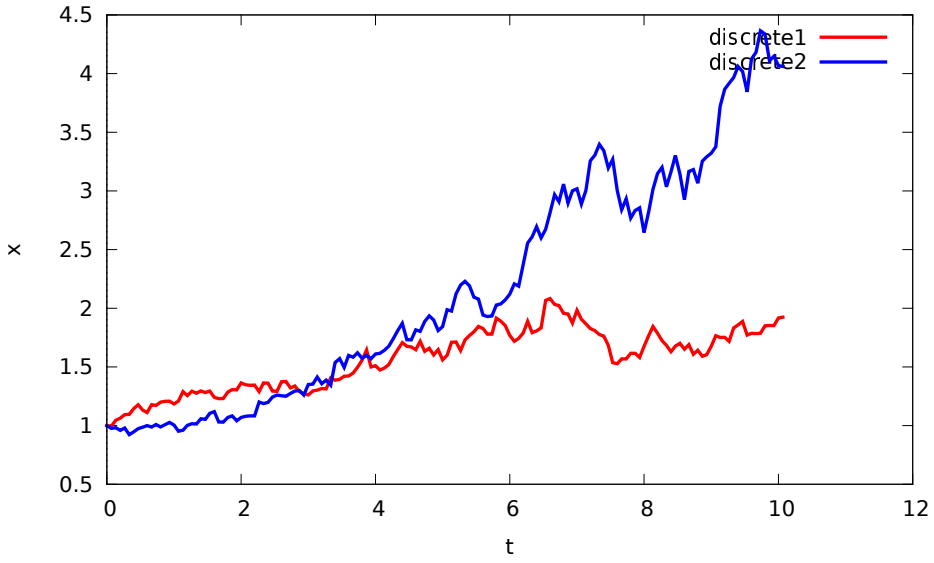

Figure 18.1: Two instances of geometric random walk

19 Linear programming: the simplex method

In linear programming, one intends to solve the following progblem. Given a vector of variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, given an $m \times 1$ column vector $\mathbf{b}$, given an $n \times 1$ column vector $\mathbf{c}$, and and $m \times n$ matrix, all with real entries, find $\mathbf{x}$ such that $\mathbf{x} \geq 0$ (meaning that $x_1 \geq 0$, $x_2 \geq 0$, …, $x_n \geq 0$), $A\mathbf{x} \leq \mathbf{b}$, and $\mathbf{c}^T x$ is the largest possible. There are various methods of finding such an $\mathbf{x}$; a frequently used and efficient such method is the simplex method.

19.1 Tableau representation of the problem

To solve this problem, first all inequalities are replaced by equations, except for the assumption that $\mathbf{x} \geq 0$. This assumption will always be made, but it will not mentioned explicitly. One introduces slack variables z and $\mathbf{s} = (s_1, s_2, \ldots, s_m)$, and assumes $\mathbf{s} \geq 0$; z will be allowed to be any real number. The problem will be reformulated as follows,

$$\begin{aligned} z - \mathbf{c}^T\mathbf{x} &= 0 \\ A\mathbf{x} + \mathbf{s} &= \mathbf{b} \end{aligned} \tag{19.1}$$

One wants to solve these equations in such a way that all variables except z are nonnegative, and z is the largest possible. Writing I for the $m \times m$ identity matrix, one represents this problem with the following matrix T, called the tableau representation of the problem:

$$T = \begin{pmatrix} 1 & -\mathbf{c}^T & 0_{1\times m} & 0 \\ 0_{m\times 1} & A & I & \mathbf{b} \end{pmatrix}; \tag{19.2}$$

here $0_{k\times l}$ indicates the $k \times l$ zero matrix. Note that the rows of the matrix T are linearly independent. This is because the only nonzero entry of the first column is in the first row, and the identity matrix occurs in the rest of the rows.

19.2 Allowed manipulations of the tableau

The following manipulations of the tableau are allowed: elementary row operations (interchange of two rows, multiplying a row by a nonzero real, and adding a multiple of a row to another row), with the following expections. One is not allowed to interchange the first row with any other row, one is not allowed to multiply the first row by any number, and one is not allowed to add a multiple of the first row to another row (but it is allowed to add a multiple of another row to the first row). These restrictions ensure that in the equations corresponding to the tableau, the variable z to be maximized will occur only in the first equation, and will occur there with coefficient one. These manipulations will result in equations equivalent to those described by the original tableau.

Further, one is allowed to interchange any two columns except that the first and the last column cannot be moved. Column interchanges amount to interchanging variables, still resulting in equivalent equations. While one needs to keep track of column interchanges so as to know which column corresponds to which variable, one does not need to keep track of row operations. When working out an example by hand, it is best not to do any column interchanges. The description of the method is simplified if one performs such column interchanges take place; as for the actual calculations, they are not helped by column interchages.[19.1]

19.3 Existence of a basic feasible solution

We are making a detour for some theoretical considerations. We will consider the linear programming problem with equations, without inequalities, as given in formula (19.1), but here we will not

[19.1] These comments apply to working out numerical examples to help one understand the method. The situation with working out examples on computer is totally different. There, one has to give special considerations as to how these matrices are represented, especially because the matrices occurring in practice are very large sparse matrices (matrices with relatively few nonzero entries); representation of such matrices for efficient calculations is a science unto itself.

distingues between the original and the slack variables. That is, we will consider the problem in the form

$$\begin{aligned} &\text{maximize:} && \mathbf{c}^T\mathbf{x} \\ &\text{subject to:} && A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq 0. \end{aligned} \tag{19.3}$$

Definition 19.1. A *basic solution* of this problem is a vector $\mathbf{x}$ satisfying the equations such that the columns of A nonzero entries of $\mathbf{x}$ are linearly independent. A *feasible solution* is one which satisfies the equation and also satisfies the requirement that $\mathbf{x} \geq 0$. A basic feasible solution is a basic solution that is also feasible. An optimal solution is a feasible solution $\mathbf{x}$ for which $\mathbf{c}^T\mathbf{x}$ is the largest possible.

Theorem 19.1. *If the problem in (19.3) has an optimal solution then it also has an optimal solution that is also a basic solution.*

Proof. Assume the size of A is $m \times n$. Let $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)^T$ be a solution that is optimal and is not basic; that is, assume that the columns of A associated with the nonzero entries of x^* are linearly dependent. Let $\mathbf{d} = (d_1, d_2, \ldots, d_n)^T \neq 0$ be a column vector such that $d_k = 0$ for all k with $1 \leq k \leq n$ for which $x_k^* = 0$ for which $A\mathbf{d} = 0$. There is such a $\mathbf{d}$, since the columns associated with the nonzero entries of $\mathbf{x}^*$ are linearly dependent.

We may assume that $\mathbf{c}^T\mathbf{d} \geq 0$; indeed, if $\mathbf{c}^T\mathbf{d} < 0$, we can replace $\mathbf{d}$ by $-\mathbf{d}$. We cannot have $\mathbf{c}^T\mathbf{d} > 0$, because then picking a small enough $\theta \geq 0$, the vector $\mathbf{x}^* + \theta\mathbf{d} \geq 0$, so it is a feasible solution, and yet

$$\mathbf{c}^T(\mathbf{x}^* + \theta\mathbf{d}) = \mathbf{c}^T\mathbf{x}^* + \mathbf{c}^T\theta\mathbf{d} > \mathbf{c}^T\mathbf{x}^*,$$

contradicting the optimatily of $\mathbf{x}^*$.

So we must have $\mathbf{c}^T\mathbf{d} = 0$. There is a small positive or negative θ such that $\mathbf{x}^* + \theta\mathbf{d} \geq 0$ and $x_k^* + \theta d_k = 0$ for at least one k for which $x_k^* > 0$. That is, $\mathbf{x}^* + \theta\mathbf{d}$ is an optimal solution that has at least one fewer nonzero entries than $\mathbf{x}^*$. Repeating this process as long as necessary, we can arrive at an optimal solution $\mathbf{x}^{**}$ such that the column vectors associated with its nonzero entries are linearly independent. In other words, $\mathbf{x}^{**}$ is an optimal basic solution. □

19.4 Bases and degeneracy

We will consider the linear programming problem as described in (19.3). We may assume that the rank of A equals the rank of the *augmented* matrix (A, b) of the given system of equations; indeed, if this is not the case, then the system of equations is not solvable. Then we may also assume that the rows of A are linearly independent; indeed, if this is not the case, we can drop some equations from the system without changing the set of solutions. So assume that A is an $m \times n$ matrix with linearly independent rows. We also assume that $m \geq n$.[19.2]

A *basic solution* of the linear programming problem given in (19.3) *associated to a basis* is a basic is a basic solution $\mathbf{x}$ of the problem such that m variables are selected, called basic variables, such that the columns of A associated with these variables are linearly independent, and all entries of $\mathbf{x}$ among the nonbasic variables are 0. The basis $\mathcal{B}$ associated to a basic solution is the list of colums of A associated with the basic variables. [19.3] Observe that given a basis $\mathcal{B}$ of the column space of A (that is, $\mathcal{B}$ consists of m linearly independent column vectors of A), a basic solution associated with $\mathcal{B}$ is uniquely determined. A basis is called *feasible* if the basic solution associated with it is

[19.2]This is certainly true in case a linear programming problem in terms of equations that arises from a linear programming problem given with inequalities by adding slack variables.

[19.3]The column vectors of $\mathcal{B}$ form a basis of the column space of A.

feasible. A basic solution is called *degenerate* if some of its basic variables are 0. A basis is called *optimal* if it is associated with an optimal basic solution.

Degenerate basic feasible solutions will cause some trouble in some of the theoretical discussion below. They also cause some trouble with the simplex algtorithm discussed below. After this discussion we will describe how to change a linear programming problem slightly such that it will have no degenerate bases and its, if it has an optimal solution, the modified problem will have its optimal solution at a feasible basis which also gives an optimal solution of the original problem.

19.5 Transforming a basis to the columns of the identity matrix

Given a basis $\mathcal{B}$, let $A_{\mathcal{B}}$ be the submatrix of A consisting of the column-vectors of $\mathcal{B}$; clearly, the matrix $A_{\mathcal{B}}$ is invertible. The matrix $A_{\mathcal{B}}^{-1}A$ transforms the columns of A corresponding to $\mathcal{B}$ to the columns of the identity matrix.[19.4]

Below, we will face the situation that the columns corresponding to a basis to columns of an identity matrix in the tableau

$$T = \begin{pmatrix} 1 & -\mathbf{c}^T & b_0 \\ 0_{m\times 1} & A & \mathbf{b} \end{pmatrix}; \tag{19.4}$$

of a linear programming problem. This tableau corresponds to the form of the linear programming problem described in formula (linprog: starting tableau) rather than in (linprog: lin prog in equ form); that is, the slack variables have been added. In the initial form given in formula (linprog: starting tableau), $b_0 = 0$ in the top right entry of the tableu, but after some manipulation described below, b_0 may change. Let $\mathbf{c}_{\mathcal{B}}^T$ be the row vector containing only the etries of $\mathbf{c}^T$ that corresponds to the columns of $\mathcal{B}$. The matrix

$$T_{\mathcal{B}} = \begin{pmatrix} 1 & -\mathbf{c}_{\mathcal{B}}^T \\ 0_{m\times 1} & A_{\mathcal{B}} \end{pmatrix};$$

is invertible. In the tableau $T_{\mathcal{B}}^{-1}T$, the leftmost column and the columns corresponding to $\mathcal{B}$ will be the columns of the $(m+1)\times(m+1)$ identity matrix.

19.6 Feasible solutions, basic and nonbasic variables

A *basic feasible solution* of equations of the tableau is an assigment of values to the variables satisfying the equations represented by the tableau such that all variables other than z are nonnegative, and all but m of them (other than z) are zero. The *basic variables* must correspond to linearly independent columns of the tableau T.[19.5] The variable z itself can assume any real value.

The solution of the equations represented by the tableau such that all variables other than z are nonnegative are represented by the points in a *convex polytope* in the $n+m+1$-dimensional euclidean space.[19.6] It follows from the linearity of the problem that the objective function (that is, the variable z) is maximized at a vertex of the polynope.[19.7]

[19.4] Multiplying a matrix by an invertible square matrix on the left corresponds to performing elementary row operations. These elementary row operations have the same effect on the columns of $A_{\mathcal{B}}$ whether or not these columns are a part of a matrix with more columns.

[19.5] We pointed out above that the rank of T is $m+1$.

[19.6] A region in an Euclidean space is a region that along with any two points it contains also contains all points of the line segment connecting the point. A polytope is the higher dimensional analog of a polygon (in two dimensions) or the polyhedron (in three dimensions.

[19.7] In some cases, called *degenerate*, such a maximum can occur at several vertices, and then it occurs an all convex linear combination of these vertices. Here a linear combination is called convex if all coefficients are nonnegative and they add up to 1.

There are two reasons why an optimal solution may not exist. First, is possible that the equations have no solution at all (subject to the requirement that all variables except for z are nonnegative); that is, the region described is empty. Second, it is possible that the region is unbounded, and the maximum described by the objective function does not exist (one might say that it is infinite). In this case, a basic feasible solution may still exist.

In describing a basic feasible solution, one may want to rename the variables (other than z) as $\mathbf{x} = (x_1, x_2, \ldots x_m)$, called *basic* or *dependent* variables, and $\mathbf{y} = (y_1, y_2, \ldots x_n)$, called *nonbasic* or *independent* variables. The variables in $\mathbf{x}$, may not be the same as those in $\mathbf{x}$ in the original statement of the problem; as we mentioned above, one needs to keep track of these name changes so that at the end one may express the solution in terms of the original variables. Once a basic feasible solution is found, one needs to rewrite the tableau T in the form

$$T = (I, C, \mathbf{d}), \tag{19.5}$$

where I is now an $(m+1) \times (m+1)$ identity matrix, $\mathbf{d} = (d_0, d_1, \ldots d_m)^T$ is an $(m+1) \times 1$ column vector. This is called the *canoncical form* of the tableau. In the basic feasible solution, we must have $z = d_0$ and $x_i = d_i$ for i with $1 \leq i \leq m$, and $y_j = 0$ for j with $1 \leq j \leq n$.

19.7 Pivoting

A basic feasible solution represents a vertex of the solution polytope, and a pivot operation represents a move from one vertex of the polytope to a neighboring vertex in such a way that it improves the value of the objective function (increases the value of z), or at least does not decrease it (cases when a pivot does not increase the value of the objective function are called degenerate).

Formally, a pivot operation represents moving out one of the basic variables, say x_k, and replacing it with a a nonbasic variable, y_l. In this operation, the new value of x_k will be 0 and y_l will have a positive value (except in degenerate cases, when we will also have $y_l = 0$).[19.8] When selecting y_l, we only want to select a y_l that occurs with negative coefficient in the top row in T in equation (19.5).[19.9] Namely, the value of y_l will be changed from 0 to a positive value, and when doing so, we must satisfy all equations. If y_l occurs with negative coefficient in the top row, increasing y_l will result in an increase of z, that is, in an improvement of the value of the objective function.[19.10] If the coefficient of y_l is positive in the top row, such an increase would result in a decrease in the value of z. If there is no y_l for which the coefficient in the top row is negative, the value of the objective function cannot be improved, meaning that we are already at an optimum.

Since a change in the value of y_l will result in changes of the variables in $\mathbf{x}$, the maximum change that can be made is the amount that makes one of the variables in $\mathbf{x}$ zero, while keeps all other variables nonnegative. The variable x_k occurs with nonzero coefficent only in the kth row (i.e., the kth equation represented by the tableau T in equation(19.5), where we top row is counted as row zero (similarly the left column is counted is column zero, to recognize the special status of the top row representing the objective function). The k equation at present can be described as $x_k + c_{kl} y_l = d_k$, since we will keep all other variables in $\mathbf{y}$ 0. Here we must have $d_k \geq 0$ since we have a solution for this equation with $x_k \geq 0$ and $y_l = 0$. If $c_{kl} \leq 0$ then this equation is solvable for any positive value of y_l and $x_k \geq 0$, so the value of y_l is not restriced by x_k. if $c_{kl} > 0$ then the largest value that can

[19.8] The handling of degenerate cases is complicated, and we will not discuss this here.

[19.9] We want to count the rows and the columns of T with the numbers 0, 1, 2, ..., to recognize the special status of the objective function, so we will avoid calling the top row the first row.

[19.10] A change of the value of y will also force a change of some or all of the variables in $\mathbf{x}$, but since all these variables occur with coefficient zero in the top row, these changes will not affect the value of the objective function. The values of all variables in $\mathbf{y}$ other than y_l will remain 0.

be given to y_k is d_k/c_{kl}, which choice results in $x_k = 0$. This means that the largest value that can be assigned to y_l is

$$\min\left\{\frac{d_k}{c_{kl}} : 1 \leq k \leq m \quad \text{and} \quad c_{kl} > 0\right\} \qquad l \text{ is fixed.}$$

In the pivot operaion, this value is assigned to y_l, and the values implied by this choice modifies the value of $\mathbf{x}$; the inequality $\mathbf{x} \geq 0$ will continue to hold, while we will have $x_k = 0$ for the value of k for which the minimum is assumed. In the pivot step, then y_l will be a new basic variable, and x_k will be a nonbasic variable. As a final step, one makes the columns corresponding to the variables x_k and y_l, and performs row operations to bring the tableau to a canonical form described in equation (19.5).[19.11] If this minimum does not exist (since the set involved is empty), then one may choose y_l arbitrarily large. In this case, z can be made arbitrarily large, to the optimum does not exist (since it is infinity).

19.8 Optimal bases

In the actual inplementation of pivoting, one does not usually do column interchages. Instead, when one changes to a new basis $\mathcal{B}$ of the tableau T, one makes the columns of the new basis into columns of an identity matrix by changing to the tableau $T_{\mathcal{B}}^{-1}T$, as described in Subsection 19.5. One can ascertain the optimality of a basis by noting that in an optimal basis no pivot is possible that improves the the value of the objective function. We formulate this in a lemma.

Lemma 19.1 (Optimality lemma)**.** *A feasible basis $\mathcal{B}$ of the linear programming problem with tableau T is optimal if all entries except for the last one in the tableau $T_{\mathcal{B}}^{-1}T$ are ≥ 0. Conversely, if T has no degenerate basis and the basis $\mathcal{B}$ is optimal, then there all entries of the first row of $T_{\mathcal{B}}^{-1}T$ other than the last entry are ≥ 0.*

Proof. We may assume that all columns of the basis $\mathcal{B}$ are on the left of the matrix A. Then the matrix $T_{\mathcal{B}}^{-1}T$ has the following form:

$$T_{\mathcal{B}}^{-1}T = \begin{pmatrix} 1 & 0_{1\times m} & -\mathbf{h}^T & d_0 \\ 0_{m\times 1} & I & B & \mathbf{d} \end{pmatrix}, \tag{19.6}$$

where I is the $m \times m$ identity matrix. Changing the original name $\mathbf{x}$ of the variables to

$$(z, \mathbf{x}^T, \mathbf{y}^T)^T = \begin{pmatrix} z \\ \mathbf{x} \\ \mathbf{y} \end{pmatrix},$$

where z is the variable representing the objective function, $\mathbf{x}$ is the column vector of basic variables, and $\mathbf{y}$ is the column vector of nonbasic variables, the linear programming problem is described by the equation

$$\begin{pmatrix} 1 & 0_{1\times m} & -\mathbf{h}^T \\ 0_{m\times 1} & I & B \end{pmatrix} \begin{pmatrix} z \\ \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} d_0 \\ \mathbf{d} \end{pmatrix}. \tag{19.7}$$

[19.11] These row operations amount to making all coefficients of the entering variable y_l 0 in rows other than row k, and making it 0 in all other rows (including row 0). This is possible, since the existence of the above minimum implies that $c_{kl} \neq 0$ for the selected k.

It is possible that there are several ks that can be selected, so there may be several choices for the variable x_k exiting the basic variables. The selection of y_l is more problematic. Any choice of y_l will do for which $c_{0l} < 0$. One may be tempted to choose y_l such that it will result in the largest possible increase in the value of z, but it may be too time-consuming to calculate the best value of l rather than pick one that is suitable.

At the basic feasible solution corresponding to the basis $\mathcal{B}$ we have $\mathbf{x} \geq 0$ and $\mathbf{y} = 0$, so we have $z = d_0$ according to these equations. At any other feasible solution described by the vector $(z', \mathbf{x'}^T, \mathbf{y'}^T)^T$ we have $\mathbf{x'} \geq 0$ and $\mathbf{y'} \geq 0$, so the equation described by the first row of the tableau becomes $z' - \mathbf{h}^T\mathbf{y'} = d_0$. Since we have $-\mathbf{h}^T \geq 0$ according to our assumptions, this shows that $z' \leq z$, showing that the solution the basic solution corresponding to the basis $\mathcal{B}$ is indeed optimal. This establishes the first assertion of the lemma.

To establish that the second assertion, assume that the basis $\mathbf{B}$ is optimal, and the vector $(z, \mathbf{x}^T, \mathbf{y}^T)$ is the feasible basic solution associated with this basis. According to the assumption, this solution is nondegenerate, so $\mathbf{x} > 0$ and $\mathbf{y} = 0$. According to equation (19.7), we have the equations

$$z - \mathbf{h}^T\mathbf{y} = d_0$$
$$\mathbf{x} + B\mathbf{y} = \mathbf{d}.$$

Since $\mathbf{x} > 0$, any one of the entries of $\mathbf{y}$ can be changed from 0 to a small positive number; such a change may change some or all of the values of the entries of the vector $\mathbf{x}$, but each entry of $\mathbf{x}$ has some room to decrease, and there is no limit how much a component of $\mathbf{x}$ is allowed to increase if the second equation here is to be satisfied. If the entry of $-\mathbf{h}^T$ correspoinding to an entry of $\mathbf{y}$ is negative, the increase of this entry will result in an increase of the value of z according to the first of the two equations above, showing that the basic solution corresponding to the basis $\mathcal{B}$ is not optimal. □

19.9 Finding a feasible basic solution.

Often one can find a basic feasible solution by the nature of the problem, but sometimes this is not the case. Assume we start with a tableau

$$T = (A, \mathbf{b}),$$

which does not need to be in canonical form, but the only nonzero entry in the first column must be 1 (corresponding to the fact that the first column represents the variable having the value of the objective function), and all entries of $\mathbf{b} = (b_0, b_1, \ldots, b_m)$ other than b_0 are nonnetvative (b_0 may be negative). If this is not already the case, one can multiply some rows of T by -1. Then one introduces new variables $\mathbf{u} = (u_1, u_2, \ldots, u + m)^T$, and replaces the objective with the requirement that the value of

$$\sum_{k=1}^{m} u_k$$

be minimized, or, equivalently, the value of $-\sum_{k=1} u_k$ be maximized. A basic feasible solution of this problem is $u_k = b_k$ for k with $1 \leq k \leq m$, and all other variables are 0 (except for the variable z describing the value of the objective function). Then one solves this linear probramming problem. If there is a basic feasible solution for which $u_1 = u_2 = \ldots u_m = 0$, then one can drop these variables can be dropped and one found a basic feasible solution of the original problem.[19.12] If no such basic solution can be found, then the original problem has no basic feasible solution, and so it is unsolvable.

[19.12]There are some issues here with the degenerate situation where some among u_k are basic variables in spite of having value 0; we will not deal with this here.

19.10 The Vandermonde matrix

We need to make a small detour in order to show that a matrix considered below is nonsingular. Let $n \geq 0$ be an integer, and let x_0, x_1, x_2, …, x_n be complex numbers. The Vandermonde matrix is the $(n+1) \times (n+1)$ matrix

$$V(x_0, x_1, \ldots, x_n) = (x_k^l)_{0 \leq k,l \leq n}, \tag{19.8}$$

where the superscript indicates exponents; we take $x_k^0 = 1$ even if $x_k = 0$. We will show by induction on n that

$$\det V(x_0, x_1, \ldots, x_n) = \prod_{k,l:0 \leq k < l \leq n} (x_l - x_k). \tag{19.9}$$

The formula is true in case $n = 0$, since then left-hand is the determinant of a 1×1 matrix with its only entry being 1, and the right-hand side is the empty product. If $x_k = x_l$ for any $k \neq l$, then both sides are zero, so we may assume that the x_ks are distinct. Assume $n > 0$, and assume equation (19.9) is true with $n-1$ replacing n, and consider the determinant $P(x) = \det V(x_0, x_1, \ldots, x_{n-1}, x)$. This is a polynomial of x of degree n; it has zeros at x_0, x_1, x_2, …, x_{n-1}. With the assumption that the x_ks are distinct, these account for all of the zeros of $P(x)$. Writing a for the leading coefficient of $P(x)$, this means that

$$P(x) = a \prod_{j=0}^{n-1} (x - x_j).$$

On the other hand, expanding the determinant $\det V(x_0, x_1, \ldots, x_{n-1}, x)$ with respect to its last column, we can see that $a = \det V(x_0, x_1, \ldots, x_{n-1})$. Writing $x = x_n$, according to the induction hypothesis, this means that

$$\det V(x_0, x_1, \ldots, x_n) = \Big(\prod_{k,l:0 \leq k < l \leq n-1} (x_l - x_k) \Big) \prod_{j=0}^{n-1} (x_n - x_j) = \prod_{k,l:0 \leq k < l \leq n} (x_l - x_k)$$

, establishing equation (19.9). What will be important for us is that this determinant is not zero if x_0, x_1, x_2, …, x_n are distinct, and so the corresponding Vandermonde matrix is nonsingular.

19.11 Elimination of degeneracy

In the discussion of the simplex method we already saw that degeneracy causes some trouble if we want to replace a basic variable that is zero with a nonbasic variable.

Theorem 19.2. *Assume the linear programming problem*

$$(\mathcal{P}) \qquad \begin{array}{ll} \textit{maximize:} & \mathbf{c}^T \mathbf{x} \\ \textit{subject to:} & A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq 0, \end{array}$$

has a feasible solution. Given an arbitrary $\epsilon > 0$, there is a vector $\mathbf{b}'$ such that $\|\mathbf{b}' - \mathbf{b}\|_\infty < \epsilon$ for which the linear programming problem

$$(\mathcal{P}') \qquad \begin{array}{ll} \textit{maximize:} & \mathbf{c}^T \mathbf{x} \\ \textit{subject to:} & A\mathbf{x} = \mathbf{b}', \quad \mathbf{x} \geq 0, \end{array}$$

also has a feasible solution. Furthermore, all basic solutions of $(\mathcal{P}')$ are nondegenerate, and every nonfeasible basis of $(\mathcal{P})$ is also a nonfeasible basis of $(\mathcal{P}')$. Finally, if $(\mathcal{P}')$ has an optimal basis $\mathcal{B}$, then $\mathcal{B}$ is also an optimal basis of $(\mathcal{P})$.

Proof. Assuming the other assumptions of the theorem are true, we first establish the assertion in the last sentence. Writing T for the tableau of the linear programming problem $(\mathcal{P})$, the tableau T' of $(\mathcal{P}')$ differs only in its last column. Assume that $\mathcal{B}$ is an optimal basis of $(\mathcal{P}')$. Since $(\mathcal{P}')$ has no degenerate basis, then all entries in the first row of ${T'_{\mathcal{B}}}^{-1}\mathcal{T}'$ other than the last are ≥ 0 acording to the second assertion of the Optimality Lemma 19.1. There entries are the same as the corresponding entries of $T_{\mathcal{B}}^{-1}\mathcal{T}$. Since the basis $\mathcal{B}$ is a feasible basis of $(\mathcal{P})$ (since every nonfeasible basis of $(\mathcal{P})$ is also a nonfeasible basis of $(\mathcal{P}')$ according to yet to be established assertions of the theorem to be proved), the basis is an optimal basis of the problem $(\mathcal{P})$ according to the first assertion of the Optimality Lemma 19.1.

We now turn to the construction of $(\mathcal{P}')$, before establishing its other properties. Let $\mathcal{C}$ be a feasible basis of $(\mathcal{P})$, and assume, for the sake of simplicity, that the columns of $\mathcal{C}$ are the leftmost columns of A. Writing T for the tableau of $(\mathcal{P})$, the tableau of $T_{\mathcal{C}}^{-1}T$ can be written similarly as in formula (19.7), that is

$$T_{\mathcal{C}}^{-1}T = \begin{pmatrix} 1 & 0_{1\times m} & -\mathbf{h}^T & d_0 \\ 0_{m\times 1} & I & B & \mathbf{d} \end{pmatrix}, \tag{19.10}$$

Again separating the variables into basic variables $\mathbf{x}$ and nonbasic variables $\mathbf{y}$ as in equation (19.7),[19.13] at the basis $\mathcal{C}$ we have $\mathbf{x} = \mathbf{d}$ and $\mathbf{y} = 0$.

To define $\mathbf{b}'$ so as to specify $(\mathcal{P}')$, we will take and appropriate $\delta > 0$ and put $\mathbf{d}' = \mathbf{d} + (\delta, \delta^2, \ldots, \delta^m)^T$. This will ensure that the basis $\mathcal{C}$ remains feasible; in fact, for the solution $({\mathbf{x}'}^T, {\mathbf{y}'}^T)^T$ of $(\mathcal{P}')$ associated with the basis $\mathcal{C}$ we have $\mathbf{x}' = \mathbf{d}'$ and $\mathbf{y}' = 0$. For the vector $\mathbf{b}'$ on the right-hand side of $(\mathcal{P}')$ we will have $\mathbf{b}' = A_{\mathcal{C}}\mathbf{d}' = \mathbf{b} + A_{\mathcal{C}}(\delta, \delta^2, \ldots, \delta^m)^T$. Assuming $\delta < 1$, this implies that $\|\mathbf{b}' - \mathbf{b}\|_\infty < \|A_{\mathcal{C}}\|_\infty \delta$. If $\mathbf{u}$ is a basic solution associated with a basis $\mathcal{B}$ of the problem $(\mathcal{P})$, and $\mathbf{u}'$ is the basic solution associated with with the same basis $\mathcal{B}$, then

$$\|\mathbf{u}' - \mathbf{u}\|_\infty \leq \|A_{\mathcal{B}}^{-1}\|_\infty \|\mathbf{b}' - \mathbf{b}\|_\infty \leq \|A_{\mathcal{B}}^{-1}\|_\infty \|A_{\mathcal{C}}\|_\infty \delta;$$

Given that there are only finitely many bases, and for an unfeasible basis $\mathcal{B}$, the associated solution $\mathbf{u}$ has a negative entry, by choosing δ small enough, we can ensure that $\mathbf{u}'$ will also have a negative entry, ensuring that no feasible unfeasible basis will be converted to a feasible basis by passing from the problem $(\mathcal{P})$ to the problem $(\mathcal{P}')$.

We have yet to show that $(\mathcal{P})$ has no degenerate basis. This is equivalent to saying that for any basis $\mathcal{B}$, the vector

$$\mathbf{p}(\delta) \stackrel{def}{=} A_{\mathcal{B}}^{-1} A_{\mathcal{C}} \big(\mathbf{d} + (\delta, \delta^2, \ldots, \delta^m)^T\big)$$

has no zero entry. To establish this, we will show that each entry of this vector is a nonconstant polynomial of δ. Since a polynomial can only have finitely many zeros, and there are only many choices for $\mathcal{B}$, it will then be possible to choose a small $\delta > 0$ such that none of these vectors has a zero entry. It is clear that each entry of the vector $\mathbf{p}(\delta)$ is a polynomial of δ. In order to show that none of the entries is a constant polynomial, write $d_0 = 0$ and $\mathbf{d} = (d_1, d_2, \ldots, d_m)^T$ Let $\delta_0 = 0$, δ_1, δ_2, $\ldots$, δ_m, be distinct real numbers and consider the matrix $(m+1) \times (m+1)$ matrix entry at the intersection of row k and column l $(0 \leq k, l \leq m)$ is $d_k + \delta_l^k$ (we take $\delta_0^0 = 1$).[19.14] The matrix M is nonsingular, because by elementary row operations (subtracting d_k times the top row from

[19.13] This equation holds at present, with the the matrix B, the vectors $\mathbf{h}$ and $\mathbf{d}$, and the quantity d_0 with new values here.

[19.14] Normally, 0^0 is not defined, but when one takes a polynomial of a variable, say δ, then it is customary to define $\delta^0 = 1$ even for $\delta = 0$.

row k) the matrix M can be transformed into the Vandermonde matrix $V(\delta_0, \delta_1, \ldots, \delta_m)$, which is nonsingular – see subsection 19.10. Hence the matrix

$$\begin{pmatrix} 1 & 0_{1\times m} \\ 0_{m\times 1} & A_{\mathcal{B}}^{-1} A_{\mathcal{C}} \end{pmatrix} M = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ \mathbf{p}(\delta_0) & \mathbf{p}(\delta_1) & \mathbf{p}(\delta_2) & \ldots & \mathbf{p}(\delta_m) \end{pmatrix}$$

is also nonsingular. So it can only have at most one constant row; this is the first row, showing that none of entries of the column vector $\mathbf{p}(\delta)$ is a constant polynomial. This completes the proof. □

20 Linear programming: duality

In the preceding chapter, we discussed the linear programming problem

$$\begin{aligned} &\text{maximize:} && \mathbf{c}^T\mathbf{x} \\ &\text{subject to:} && A\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{20.1}$$

We will call this the primal problem. The dual of this problem will be defined as

$$\begin{aligned} &\text{minimize:} && \mathbf{b}^T\mathbf{y} \\ &\text{subject to:} && A^T\mathbf{y} \geq \mathbf{c}. \end{aligned} \tag{20.2}$$

Here A is a given matrix, $\mathbf{b}$ and $\mathbf{c}$ are given column vectors, and $\mathbf{x} \geq 0$ and $\mathbf{y} \geq 0$ are column vectors to be determined. It is assumed that these vectors are of appropriate sizes so that the indicated multiplications can be carried out, and the indicated equations and inequalities make sense. A feasible solution for the primal problem is a vector ≥ 0 that satisfies the required inequalities (whether or not it achieves the optimum); similarly for the dual problem. It is easy to see that the dual of the dual problem is the primal problem.

The weak duality theorem asserts that the for any feasible solution of the primal problem, the value of the objective function is less than or equal to the value of the objective function for any feasible solution of the dual problem. That is

Theorem 20.1 (Weak duality theorem)**.** *Assume that* $\mathbf{x}$ *is a feasible solution of the primal problem (20.1) and* $\mathbf{y}$ *is a feasible solution of the dual problem (20.2). Then* $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$.

The proof is immediate:

Proof. We have

$$c^T\mathbf{x} \leq (A^T\mathbf{y})^T\mathbf{x} = (\mathbf{y}^T A)\mathbf{x} = \mathbf{y}^T(A\mathbf{x}) \leq \mathbf{y}^T\mathbf{b} = (\mathbf{y}^T\mathbf{b})^T = \mathbf{b}^T\mathbf{y};$$

here, the first inequality follows from the inequality in formula (20.2), and the second inequality follows form the inequality in formula (20.1). Finally, the penultimate[20.1] equation follows since $\mathbf{y}^T\mathbf{b}$ is a scalar, so it is equal to its own transpose. □

The strong duality theorem asserts that if an optimum for the primal problem exists, the dual problem also has an optimum, and the two optima are the same.

[20.1] The one before the last one

Theorem 20.2 (Strong duality theorem)**.** *Assume that* $\mathbf{x}$ *is an optimal solution of the primal problem (20.1). Then there is a* $\mathbf{y}$ *that is an optiman solution of the primal problem (20.2). Furthermore we have*

$$\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}. \tag{20.3}$$

In the proof, we will assume that the primal problem has no degenerate basic feasible solution. What this means that in a basic feasible solution, none of the basic variables is 0. When there is a degenerate basic feasible solution, we can apply Theorem 19.2 with $\epsilon \searrow 0$.

Proof. We start with the original tableau of the primal problem as described in formula (19.2), and we will make pivots without making column interchanges. This will only affect pivoting in that in a basic feasible solution, the basic variables will stay in place, rather than moved to as shown in equation (19.5). Then the new tableu can be obtained from the original tablea by row operations; since row operations on a matrix correspond to multiplying the matrix by a nonsingular matrix on the left, after performing a number of pivots the new tableau will look like

$$\begin{aligned} T' &= \begin{pmatrix} 1 & \mathbf{y}^T \\ 0 & M \end{pmatrix} T = \begin{pmatrix} 1 & \mathbf{y}^T \\ 0 & M \end{pmatrix} \begin{pmatrix} 1 & -\mathbf{c}^T & 0_{1\times m} & 0 \\ 0_{m\times 1} & A & I & \mathbf{b} \end{pmatrix} \\ &= \begin{pmatrix} 1 & -\mathbf{c}^T + \mathbf{y}^T A & \mathbf{y}^T & \mathbf{y}^T\mathbf{b} \\ 0_{m\times 1} & MA & M & M\mathbf{b} \end{pmatrix}. \end{aligned}$$

As in formula (19.2), T has $m+1$ rows, so M is an $m \times m$ square matrix, and $\mathbf{y}$ is a $m \times 1$ column vector. Here M represents the row operations on rows 1 through m of the matrix T (the top row being row 0), and the vector $\mathbf{y}$ represents a addition of multiples of row 1 through m from row 0. Assuming the tableau represents the optimal feasible solution of the problem, all entries of the top row except for the its leftmost entry (which stays 1) and its rightmost entry must be ≤ 0. This is because the entry over a basic variable must be 0, and the entry over a nonbasic varialble must be nonnegative. The reason for the latter is that the nonbasic variables are currently 0, but any one of them can be made to be a small positive value. Indeed, such a change would involve only a small change in the values of the basic variables; since, they are currently all positive, an small enough change would not make them negative, so the solution would still remain feasible, (though probably not a basic feasible solution). If the entry in the matrix on the right-hand side over one of the nonbasic variable is negative, then we can make this variable slightly positive while increasing the value of the objective solution, meaning that the solution represented by the tableau T' is not optimal.

Hence $-\mathbf{c}^T + \mathbf{y}^T A \geq 0$ and $\mathbf{y}^T \geq 0$. This means that $\mathbf{y}$ is a solution of the dual problem. The first row of tableau T' means that

$$z + (-\mathbf{c}^T + \mathbf{y}^T A)\mathbf{x} + \mathbf{y}^T\mathbf{s} = \mathbf{y}^T\mathbf{b},$$

where $\mathbf{x}$ are the main variables of the primal problem, and $\mathbf{s}$ are its slack variables. Actually, the terms on the left-hand side other than z are 0. This is because the coefficents of the basic variables are 0, and the nonbasic variables themselves are 0, that is,

$$z = \mathbf{y}^T\mathbf{b}, \tag{20.4}$$

Here z represents the value of the objective function for the variables $(\mathbf{x}^T, \mathbf{s}^T)^T$ at the optimamum point of the the primal problem, that is $z = c^T\mathbf{x}$ (cf. (19.1) and (19.5)), while $\mathbf{y}$ is a feasible

solution of the dual problem. Noting that $\mathbf{y}^T\mathbf{b}$ being a scalar, it is equal to its own dual, that is $\mathbf{y}^T\mathbf{b} = (y^T\mathbf{b}) = \mathbf{b}^T\mathbf{y}$. Hence, equation (20.4) implies that

$$\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}.$$

$\mathbf{y}$ gives the optimum solution of the dual problem according to the weak Duality Theorem, i.e., Theorem 20.1. for the places $\mathbf{x}$ and $\mathbf{y}$ of optimum of the primal and dual problem, respectively. The last displayed equation confirms equation (20.3). □

21 Finding the maximum of a function

We want to find the maximum of a functiono f on an interval $[a, b]$ by numerial search. We will assume that the function is continuous, had a single maximum at $c \in [a, b]$, and f is strictly increasing in the interval $[a, c]$ and strictly decreasing in the interval $[c, b]$. Such a function will be called *unimodal*.[21.1]

21.1 A review of bisection finding zeros

Let f be a continuous function on the interval $[a, b]$, and $f(a)f(b) < 0$.[21.2] Then, by the Intermediate Value Theorem, the equation $f(x) = 0$ must have at least one solution in the interval (a, b). Such a solution may be approximated by successive halving of the interval. Namely, writing $x_1 = a$ and $x_2 = b$, if $x_3 = (x_1 + x_2)/2$, and $f(x_1)f(x_3) < 0$ then one of these solutions must be in the interval (x_1, x_3), and if $f(x_1)f(x_3) > 0$, then there must be a solution in the interval (x_3, x_2); of course if $f(x_1)f(x_3) = 0$ then x_3 itself is a solution. By repeating this halving of the interval, a root can be localized in successively smaller intervals. There is no guarantee that all roots are found this way, since both of the intervals (x_1, x_3) and (x_3, x_2) may contains roots of the equation $f(x) = 0$.

21.2 Golden mean search

This method will not work for finding the maximum, since we need to at least two points in an interval to localize the maximum. We will assume that f is a continuous function on the interval $[a, b]$ that is unimodal in the sense described at the beginning of this section. We are going to define a sequence of closed intervals

$$[a, b] = I_0 \supset I_1 \supset I_2 \supset I_3 \supset \dots$$

such that, for every $n \geq 0$, the place c of maximum of f on $[a, b]$ is contained in I_n, and $|I_n|/|I_{n+1}| = (1 + \sqrt{5})/2$ (the number on the right-hand side is called the *golden ratio* or the *golden mean*), where I denotes the length of the interval I.

Assume I_n has already been constructed with $c \in I_n$, and write $I_n = [x_1, x_4]$. We will pick points x_2 and x_3 with $x_1 < x_2 < x_3 < x_4$ to localize c. Clearly, we have either $c \in [x_1, x_3]$ or $c \in [x_2, x_4]$ According to the assumptions on f, if $f(x_2) \geq f(x_3)$, then we must have $c \in [x_1, x_3]$, and if $f(x_2) \leq f(x_3)$, then we must have if $c \in [x_2, x_4]$. Unless $f(x_2) = f(x_3)$, In the former case, we specify $I_{n+1} = [x_1, x_3]$, in the latter case, $I_{n+1} = [x_2, x_4]$. If $f(x_2) = f(x_3)$, we can make either choice.

As for the selection of x_2 and x_3, we need to calculate the function values at x_2 and x_3; since function evaluations may be expensive, we want to use both of these points. That is, if we pick

[21.1] The term *unimodal* comes from statistics, but since its meaning is not quite agreed upon, it should not be used without first defining it.

[21.2] This is just a simple way of saying that $f(a)$ and $f(b)$ have different signs.

$I_{n+1} = [x_2, x_4]$, and in the next step, we pick two points inside I_{n+1}, we want to make sure that one of these points will be x_3, since f has alerady been evaluated at x_3. Similarly, if we pick $I_{n+1} = [x_1, x_3]$, and in the next step, we pick two points inside I_{n+1}, we want to make sure that one of these points will be x_2, since f has alerady been evaluated at x_2. Next, we also want to make sure that $|I_n|/|I_{n+1}| = |I_{n+1}|/|I_{n+2}|$. The restriction that the length of I_{n+1} does not depend on whether we pick x_2 or x_3 as one of its endpoints implies that we must have

$$x_3 - x_1 = x_4 - x_2,$$

and the equlity of the above ratios implies that such that

$$\frac{x_4 - x_1}{x_3 - x_1} = \frac{x_3 - x_1}{x_3 - x_1}$$

Writing $d = x_4 - x_1$ and $a = x_3 - x_1 = x_4 - x_2$, we have $x_2 - x_1 = (x_4 - x_1) - (x_4 - x_2) = d - a)$, and so this means this means that $d/a = a/(d - a)$, i.e., that $d^2 - ad = a^2$. Dividing this equation by a^2 and writing $\phi = d/a$, we have $\phi^2 - \phi = 1$, that is $\phi^2 - \phi - 1 = 0$. Noting that $\phi > 1$ and using the quadratic formula, we obtain that

$$\phi = \frac{1 + \sqrt{5}}{2}.$$

As we mentioned above, the number ϕ is called the golden ratio or golden mean. Noting that the equation $\phi^2 - \phi - 1 = 0$ implies that $\phi^{-1} = \phi - 1$. That is,

$$x_3 = x_1 + \phi^{-1} \cdot (x_4 - x_1) = x_1 + (\phi - 1)(x_4 - x_1). \tag{21.1}$$

As a consequence, we have

$$x_4 - x_1 = \phi \cdot (x_3 - x_1). \tag{21.2}$$

Similarly,

$$\begin{aligned} x_2 &= x_4 - \phi^{-1} \cdot (x_4 - x_1) = x_1 + (1 - \phi^{-1})(x_4 - x_1) \\ &= x_1 + (1 - \phi^{-1})\, \phi \cdot (x_3 - x_1) = x_1 + (\phi - 1)(x_3 - x_1). \end{aligned} \tag{21.3}$$

One can make the following conclusions from equations (21.1 and (21.3). Given $I_n = [x_1, x_4]$, if one picks $I_{n+1} = [x_1, x_3]$, and writing $x_{1\,\text{new}} = x_1$ and $x_{4\,\text{new}} = x_3$, one needs to pick $x_{3\,\text{new}} = x_2$ and

$$x_{2\,\text{new}} = x_{1\,\text{new}} + (1 - \phi^{-1})(x_{4\,\text{new}} - x_{1\,\text{new}}).$$

Similarly, if one pick $I_{n+1} = [x_2, x_4]$, then one picks $x_{1\,\text{new}} = x_2$ and $x_{4\,\text{new}} = x_4$, $x_{2\,\text{new}} = x_3$, and

$$x_{3\,\text{new}} = x_{1\,\text{new}} + (\phi - 1)(x_{4\,\text{new}} - x_{1\,\text{new}}).$$

See [9, Section 10.1, p 401] for more details.

References

[1] Frank R. Giordano, William P. Fox, and Steven B. Horton. *A First Course in Mathematical Modeling*. Brooks/Cole (Cengage Learning), Boston, Massachusetts, fifth edition, 2014.

[2] Attila Máté. Introduction to numerical analysis with C programs. http://www.sci.brooklyn.cuny.edu/~mate/nml/numanal.pdf, August 2013.

[3] Attila Máté. The cyclic decomposition theorem. http://www.sci.brooklyn.cuny.edu/~mate/misc/cyclic_decomposition.pdf, December 2014.

[4] Attila Máté. The Jordan canonical form. http://www.sci.brooklyn.cuny.edu/~mate/misc/jordan_canonical.pdf, December 2014.

[5] Attila Máté. The natural exponential function. http://www.sci.brooklyn.cuny.edu/~mate/misc/exp_x.pdf, September 2015.

[6] Attila Máté. Aspects of time series, December 2016. http://www.sci.brooklyn.cuny.edu/~mate/misc/time_series.pdf.

[7] Walter J. Meyer. *Concepts of Mathematical Modeling*. Dover Publications, Minneola, New York, 2004.

[8] Edward Nelson. *Quantum Fluctuations*. Princeton Series in Physics. Princeton University Press, Princeton, New Jersey, 1985. https://web.math.princeton.edu/~nelson/books/qf.pdf.

[9] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++. The Art of Scientific Computing*. Cambridge University Press, New York, second edition, 1992.

[10] Hernann Weyl. Über die Gleichverteilung von Zahlen mod. Eins. *Mathematische Annalen*, 77(3):313–352, 1916. Free access: Mathematics Annalen.

[11] Bernard Widynski. Middle square Weyl sequence RNG. https://arxiv.org/abs/1704.00358, April 4, 2017.

[12] Wenyu Zhang. Introduction to Ito's lemma. http://www.math.cornell.edu/~web6720/Wendy_slides.pdf, May 6, 2015.