10

Solutions of the exercises

In this chapter we will provide solutions of the exercises that we have proposed at the end of the previous eight chapters. The expression "Solution n.m" is an abridged notation for "Solution of Exercise n.m" (mth Exercise of the nth Chapter).

10.1 Chapter 1

Exercise 1.1 Only the numbers of the form $\pm 0.1a_2 \cdot 2^e$ with $a_2 = 0, 1$ and $e = \pm 2, \pm 1, 0$ belong to the set $\mathbb{F}(2, 2, -2, 2)$. For a given exponent, we can represent in this set only the two numbers 0.10 and 0.11, and their opposites. Consequently, the number of elements belonging to $\mathbb{F}(2, 2, -2, 2)$ is 20. Finally, $\epsilon_M = 1/2$.

Exercise 1.2 For any fixed exponent, each of the digits $a_2, \ldots, a_t$ can assume β different values, while a_1 can assume only $\beta - 1$ values. Therefore $2(\beta-1)\beta^{t-1}$ different numbers can be represented (the 2 accounts for the positive and negative sign). On the other hand, the exponent can assume $U - L + 1$ values. Thus, the set $\mathbb{F}(\beta, t, L, U)$ contains $2(\beta - 1)\beta^{t-1}(U - L + 1)$ different elements.

Exercise 1.3 Thanks to the Euler formula $i = e^{i\pi/2}$; we obtain $i^i = e^{-\pi/2}$, that is, a real number. In MATLAB

```
exp(-pi/2)
ans =
    0.2079
i^i
ans =
    0.2079
```

Exercise 1.4 Use the instructions `U=2*eye(10)-3*diag(ones(8,1),2)` and `L=2*eye(10)-3*diag(ones(8,1),-2)`.

A. Quarteroni et al., *Scientific Computing with MATLAB and Octave*,
Texts in Computational Science and Engineering 2,
DOI 10.1007/978-3-642-45367-0_10, © Springer-Verlag Berlin Heidelberg 2014

Exercise 1.5 We can interchange the third and seventh rows of the previous matrix using the instructions: `r=[1:10]; r(3)=7; r(7)=3; Lr=L(r,:)`. Notice that the character `:` in `L(r,:)` ensures that all columns of `L` are spanned in the usual increasing order (from the first to the last). To interchange the fourth column with the eighth column we can write `c=[1:10]; c(8)=4; c(4)=8; Lc=L(:,c)`. Similar instructions can be used for the upper triangular matrix.

L(r,:)

Exercise 1.6 We can define the matrix `A = [v1;v2;v3;v4]` where `v1`, `v2`, `v3` and `v4` are the 4 given row vectors. They are linearly independent iff the determinant of `A` is different from 0, which is not true in our case.

Exercise 1.7 The two given functions f and g have the symbolic expression:

```
syms x
f=sqrt(x^2+1); pretty(f)
```

$$(x^2+1)^{1/2}$$

```
g=sin(x^3)+cosh(x); pretty(g)
```

$$\sin(x^3) + \cosh(x)$$

pretty

The command `pretty(f)` prints the symbolic expression `f` in a format that resembles type-set mathematics. At this stage, the symbolic expression of the first and second derivatives and the integral of f can be obtained with the following instructions:

```
diff(f,x)
ans =
1/(x^2+1)^(1/2)*x
diff(f,x,2)
ans =
-1/(x^2+1)^(3/2)*x^2+1/(x^2+1)^(1/2)
int(f,x)
ans =
1/2*x*(x^2+1)^(1/2)+1/2*asinh(x)
```

Similar instructions can be used for the function g.

Exercise 1.8 The accuracy of the computed roots downgrades as the polynomial degree increases. This experiment reveals that the accurate computation of the roots of a polynomial of high degree can be troublesome.

Exercise 1.9 Here is a possible program to compute the sequence:

```
function I=sequence(n)
I = zeros(n+2,1); I(1) = (exp(1)-1)/exp(1);
for i = 0:n, I(i+2) = 1 - (i+1)*I(i+1); end
```

The sequence computed by this program doesn't tend to zero (as `n` increases), but it diverges with alternating sign. This behavior is a direct consequence of rounding errors propagation.

Exercise 1.10 The anomalous behavior of the computed sequence is due to the propagation of roundoff errors from the innermost operation. In particular, when $4^{1-n}z_n^2$ is less than $\epsilon_M/2$, the subsequent element z_{n+1} of the sequence is equal to 0. This happens for $n \geq 30$.

Exercise 1.11 The proposed method is a special instance of the Monte Carlo method and is implemented by the following program:

```
function mypi=pimontecarlo(n)
x = rand(n,1); y = rand(n,1);
z = x.^2+y.^2;
v = (z <= 1);
m=sum(v); mypi=4*m/n;
```

The command `rand` generates a sequence of pseudo-random numbers. The instruction `v = (z <= 1)` is a shortand version of the following procedure: we check whether `z(k) <= 1` for any component of the vector `z`. If the inequality is satisfied for the `k`th component of `z` (that is, the point `(x(k),y(k))` belongs to the interior of the unit circle) `v(k)` is set equal to 1, and to 0 otherwise. The command `sum(v)` computes the sum of all components of `v`, that is, the number of points falling in the interior of the unit circle.

sum

By launching the program `mypi=pimontecarlo(n)` for different values of `n`, when `n` increases, the approximation `mypi` of π becomes more accurate. For instance, for `n=1000` we obtain `mypi=3.1120`, whilst for `n=300000` we have `mypi=3.1406`. (Obviously, since the numbers are randomly generated, the result obtained with the same value of `n` may change at each run.)

Exercise 1.12 To answer the question we can use the following *function*:

```
function pig=bbpalgorithm(n)
pig = 0;
for m=0:n
  m8 = 8*m;
  pig = pig + (1/16)^m*(4/(m8+1)-(2/(m8+4)+ ...
        1/(m8+5)+1/(m8+6)));
end
```

For `n=10` we obtain an approximation `pig` of π that coincides (up to MATLAB precision) with the persistent MATLAB variable `pi`. In fact, this algorithm is extremely efficient and allows the rapid computation of hundreds of significant digits of π.

Exercise 1.13 The binomial coefficient can be computed by the following program (see also the MATLAB function `nchoosek`):

nchoosek

```
function bc=bincoeff(n,k)
k = fix(k); n = fix(n);
if k > n, disp('k must be between  0 and n');
   return; end
if k > n/2, k = n-k; end
if k <= 1,  bc = n^k; else
  num = (n-k+1):n; den = 1:k; el = num./den;
  bc = prod(el);
end
```

The command `fix(k)` rounds `k` to the nearest integer smaller than `k`. The command `disp(string)` displays the string, without printing its name. The command `return` terminates the execution of the function. Finally, `prod(el)` computes the product of all elements of the vector `el`.

fix

return

prod

Exercise 1.14 The following *functions* compute f_n using the form $f_i = f_{i-1} + f_{i-2}$ (`fibrec`) or using the form (1.14) (`fibmat`):

```
function f=fibrec(n)
if n == 0
    f = 0;
elseif n == 1
    f = 1;
else
    f = fibrec(n-1)+fibrec(n-2);
end

function f=fibmat(n)
f = [0;1];
A = [1 1; 1 0];
f = A^n*f;
f = f(1);
```

For `n=20` we obtain the following results:

```
t=cputime; fn=fibrec(20), cpu=cputime-t
fn =
         6765
cpu =
    0.48
t=cputime; fn=fibmat(20), cpu=cputime-t
fn =
         6765
cpu =
     0
```

The recursive *function* `fibrec` requires much more CPU time than `fibmat`. The latter requires to compute only the power of a matrix, an easy operation in MATLAB.

10.2 Chapter 2

Exercise 2.1 The command `fplot` allows us to study the graph of the given function f for various values of γ. For $\gamma = 1$, the corresponding function does not have real zeros. For $\gamma = 2$, there is only one zero, $\alpha = 0$, with multiplicity equal to four (that is, $f(\alpha) = f'(\alpha) = f''(\alpha) = f'''(\alpha) = 0$, while $f^{(4)}(\alpha) \neq 0$). Finally, for $\gamma = 3$, f has two distinct zeros, one in the interval $(-3, -1)$ and the other one in $(1, 3)$. In the case $\gamma = 2$, the bisection method cannot be used since it is impossible to find an interval (a, b) in which $f(a)f(b) < 0$. For $\gamma = 3$, starting from the interval $[a, b] = [-3, -1]$, the bisection method (Program 2.1) converges in 34 iterations to the value $\alpha = -1.85792082914850$ (with $f(\alpha) \simeq -3.6 \cdot 10^{-12}$), using the following instructions:

```
f=@(x) cosh(x)+cos(x)-3; a=-3; b=-1;
tol=1.e-10; nmax=200;
[zero,res,niter]=bisection(f,a,b,tol,nmax)

zero =
   -1.8579
res =
    -3.6872e-12
niter =
    34
```

Similarly, choosing `a=1` and `b=3`, for $\gamma = 3$ the bisection method converges after 34 iterations to the value $\alpha = 1.8579208291485$ with $f(\alpha) \simeq -3.6877 \cdot 10^{-12}$.

Exercise 2.2 We have to compute the zeros of the function $f(V) = pV + aN^2/V - abN^3/V^2 - pNb - kNT$, where N is the number of molecules. Plotting the graph of f, we see that this function has just a simple zero in the interval $(0.01, 0.06)$ with $f(0.01) < 0$ and $f(0.06) > 0$. We can compute this zero using the bisection method as follows:

```
f=@(x) 35000000*x+401000./x-17122.7./x.^2-1494500;
[zero,res,niter]=bisection(f,0.01,0.06,1.e-12,100)

zero =
    0.0427
res =
  -6.3814e-05
niter =
    35
```

Exercise 2.3 The unknown value of ω is the zero of the function $f(\omega) = s(1, \omega) - 1 = 9.8[\sinh(\omega) - \sin(\omega)]/(2\omega^2) - 1$. From the graph of f we conclude that f has a unique real zero in the interval $(0.5, 1)$. Starting from this interval, the bisection method computes the value $\omega = 0.61214447021484$ with the desired tolerance in 15 iterations as follows:

```
f=@(omega) 9.8/2*(sinh(omega)-sin(omega))./omega.^2-1;
[zero,res,niter]=bisection(f,0.5,1,1.e-05,100)
zero =
   6.1214e-01
res =
   3.1051e-06
niter =
    15
```

Exercise 2.4 The inequality (2.6) can be derived by observing that $|e^{(k)}| < |I^{(k)}|/2$ with $|I^{(k)}| < \frac{1}{2}|I^{(k-1)}| < 2^{-k-1}(b-a)$. Consequently, the error at the iteration k_{min} is less than ε if k_{min} is such that $2^{-k_{min}-1}(b-a) < \varepsilon$, that is, $2^{-k_{min}-1} < \varepsilon/(b-a)$, which proves (2.6).

Exercise 2.5 The implemented formula is less sensitive to roundoff errors.

Exercise 2.6 In Solution 2.1 we have analyzed the zeros of the given function with respect to different values of γ. Let us consider the case when $\gamma = 2$. Starting from the initial guess $x^{(0)} = 1$, the Newton method (Program 2.2) converges to the value $\bar{\alpha} = 1.4961e-4$ in 31 iterations with `tol=1.e-10` while the exact zero of f is equal to 0. This discrepancy is due to the fact that f is almost a constant in a neighborhood of its zero, whence the root-finding is an ill-conditioned problem (see the comment at the end of Sect. 2.8.2). The method converges at the same solution and with the same number of iterations even if we set `tol=`ϵ_M. Actually, the corresponding residual computed by MATLAB is 0. Let us set now $\gamma = 3$. The Newton method with `tol=`$\epsilon_M$converges to

the value 1.85792082915020 in 9 iterations starting from $x^{(0)} = 1$, while if $x^{(0)} = -1$ after 9 iterations it converges to the value -1.85792082915020 (in both cases the residuals are zero in MATLAB).

Exercise 2.7 The square and the cube roots of a number a are the solutions of the equations $x^2 = a$ and $x^3 = a$, respectively. Thus, the corresponding algorithms are: for a given $x^{(0)}$ compute

$$x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right), \ k \geq 0 \qquad \text{for the square root,}$$

$$x^{(k+1)} = \frac{1}{3}\left(2x^{(k)} + \frac{a}{(x^{(k)})^2}\right), \ k \geq 0 \text{ for the cube root.}$$

Exercise 2.8 Setting $\delta x^{(k)} = x^{(k)} - \alpha$, from the Taylor expansion of f we find:

$$0 = f(\alpha) = f(x^{(k)}) - \delta x^{(k)} f'(x^{(k)}) + \frac{1}{2}(\delta x^{(k)})^2 f''(x^{(k)}) + \mathcal{O}((\delta x^{(k)})^3). \quad (10.1)$$

The Newton method yields

$$\delta x^{(k+1)} = \delta x^{(k)} - f(x^{(k)})/f'(x^{(k)}). \quad (10.2)$$

Combining (10.1) with (10.2), we have

$$\delta x^{(k+1)} = \frac{1}{2}(\delta x^{(k)})^2 \frac{f''(x^{(k)})}{f'(x^{(k)})} + \mathcal{O}((\delta x^{(k)})^3).$$

After division by $(\delta x^{(k)})^2$ and letting $k \to \infty$ we prove the convergence result.

Exercise 2.9 For certain values of β the equation (2.2) can have two roots that correspond to different configurations of the rods system. The two initial values that are suggested have been chosen conveniently to allow the Newton method to converge toward one or the other root, respectively. We solve the problem for $\beta = k\pi/150$ with $k = 0, \ldots, 100$ (if $\beta > 2.6389$ the Newton method does not converge since the system has no admissible configuration). We use the following instructions to obtain the solution of the problem (shown in Figure 10.1, left):

```
a1=10; a2=13; a3=8; a4=10;
ss = (a1^2 + a2^2 - a3^2+ a4^2)/(2*a2*a4);
n=150; x01=-0.1; x02=2*pi/3; nmax=100;
beta=zeros(100,1);
for k=0:100
   w = k*pi/n; i=k+1; beta(i) = w;
   f  = @(x) 10/13*cos(w)-cos(x)-cos(w-x)+ss;
   df = @(x) sin(x)-sin(w-x);
   [zero,res,niter]=newton(f,df,x01,1e-5,nmax);
   alpha1(i) = zero; niter1(i) = niter;
   [zero,res,niter]=newton(f,df,x02,1e-5,nmax);
   alpha2(i) = zero; niter2(i) = niter;
end
plot(beta,alpha1,'c--',beta,alpha2,'c','Linewidth',2)
grid on
```

The components of the vectors `alpha1` and `alpha2` are the angles computed for different values of β, while the components of the vectors `niter1` and `niter2` are the number of Newton iterations (between 2 and 6) necessary to compute the zeros with the requested tolerance.

Exercise 2.10 From an inspection of its graph we see that f has two positive real zeros ($\alpha_2 \simeq 1.5$ and $\alpha_3 \simeq 2.5$) and one negative ($\alpha_1 \simeq -0.5$). The Newton method converges in 4 iterations (having set $x^{(0)} = -0.5$ and `tol = 1.e-10`) to the value α_1:

```
f=@(x) exp(x)-2*x^2; df=@(x) exp(x)-4*x;
x0=-0.5; tol=1.e-10; nmax=100;
format long; [zero,res,niter]=newton(f,df,x0,tol,nmax)

zero =
  -0.53983527690282
res =
     0
niter =
     4
```

The given function has a maximum at $\bar{x} \simeq 0.3574$ (which can be obtained by applying the Newton method to the function f'): for $x^{(0)} < \bar{x}$ the method converges to the negative zero. If $x^{(0)} = \bar{x}$ the Newton method cannot be applied since $f'(\bar{x}) = 0$. For $x^{(0)} > \bar{x}$ the method converges to one of the two positive zeros, either α_2 or α_3.

Exercise 2.11 Let us set $x^{(0)} = 0$ and `tol`$= \epsilon_M$. In MATLAB the Newton method converges in 43 iterations to the value 0.641182985886554, while in Octave it converges in 32 iterations to the value 0.641184396264531. By taking the MATLAB approximated value as the reference solution in our error analysis, we can observe that the (approximate) errors decrease only linearly when k increases (see Figure 10.1, right). This behavior is due to the fact that α has a multiplicity greater than 1. To recover a second-order method we can use the modified Newton method.

Exercise 2.12 We should compute the zero of the function $f(x) = \sin(x) - \sqrt{2gh/v_0^2}$. By inspecting its graph, we can conclude that f has one zero in the interval $(0, \pi/2)$. The Newton method with $x^{(0)} = \pi/4$ and `tol`$= 10^{-10}$ converges in 5 iterations to the value 0.45862863227859.

Exercise 2.13 Using the data given in the exercise, the solution can be obtained with the following instructions:

```
M=6000; v=1000; f=@(r) M-v*(1+r)./r.*((1+r).^5-1);
df=@(r) v*((1+r).^5.*(1-5*r)-1)./(r.^2);
[zero,res,niter]=bisection(f,0.01,0.1,1.e-12,5);
[zero,res,niter]=newton(f,df,zero,1.e-12,100)
```

The Newton method converges to the desired result in 3 iterations.

Exercise 2.14 By a graphical study, we see that (2.38) is satisfied for a value of α in $(\pi/6, \pi/4)$. Using the following instructions:

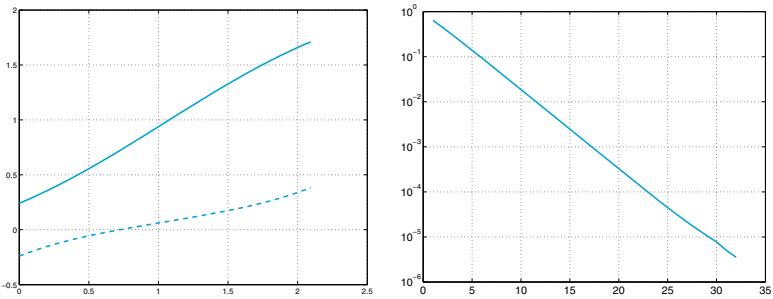

Figure 10.1. At left: the two curves represent the two possible configurations of roads system in terms of the angle α versus $\beta \in [0, 2\pi/3]$ (Solution 2.9). At right: error versus iteration number of the Newton method for the computation of the zero of the function $f(x) = x^3 - 3x^2 2^{-x} + 3x4^{-x} - 8^{-x}$ (Solution 2.11)

```
l1=8; l2=10; g=3*pi/5;
f=@(a) -l2*cos(g+a)/sin(g+a)^2-l1*cos(a)/sin(a)^2;
df=@(a) [l2/sin(g+a)+2*l2*cos(g+a)^2/sin(g+a)^3+...
         l1/sin(a)+2*l1*cos(a)^2/sin(a)^3];
[zero,res,niter]=newton(f,df,pi/4,1.e-15,100)
L=l2/sin(2*pi/5-zero)+l1/sin(zero)
```

the Newton method provides the approximate value 0.59627992746547 in 6 iterations, starting from $x^{(0)} = \pi/4$. We deduce that the maximum length of a rod that can pass in the corridor is $L = 30.5484$.

Exercise 2.15 If α is a zero of f with multiplicity m, then there exists a function h such that $h(\alpha) \neq 0$ and $f(x) = h(x)(x-\alpha)^m$. By computing the first derivative of the iteration function ϕ_N of the Newton method, we have

$$\phi_N'(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}.$$

By replacing f, f' and f'' with the corresponding expressions as functions of $h(x)$ and $(x-\alpha)^m$, we obtain $\lim_{x\to\alpha} \phi_N'(x) = 1 - 1/m$, hence $\phi_N'(\alpha) = 0$ if and only if $m = 1$. Consequently, if $m = 1$ the method converges at least quadratically, according to (2.9). If $m > 1$ the method converges with order 1 according to Proposition 2.1.

Exercise 2.16 Let us inspect the graph of f by using the following commands:

```
f=@(x) x^3+4*x^2-10; fplot(f,[-10,10]); grid on;
fplot(f,[-5,5]); grid on;
fplot(f,[0,2]); grid on; axis([0,2,-10,15])
```

we can see that f has only one real zero, equal approximately to 1.36 (see Figure 10.2, left, for the last graph generated by the previous instructions). The iteration function and its derivative are:

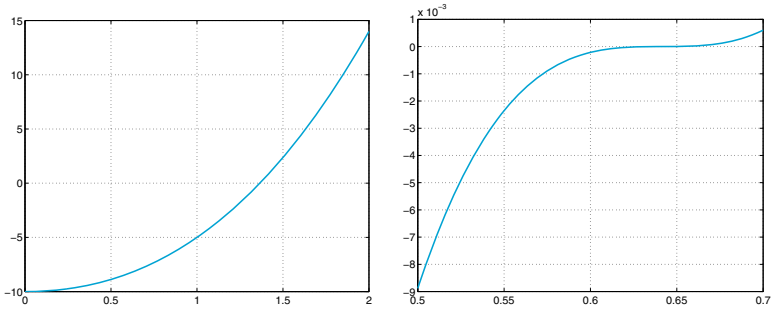

Figure 10.2. At left: graph of $f(x) = x^3 + 4x^2 - 10$ for $x \in [0, 2]$ (Solution 2.16). At right: graph of $f(x) = x^3 - 3x^2 2^{-x} + 3x4^{-x} - 8^{-x}$ for $x \in [0.5, 0.7]$ (Solution 2.18)

$$\begin{aligned}
\phi(x) &= \frac{2x^3 + 4x^2 + 10}{3x^2 + 8x} = -\frac{f(x)}{3x^2 + 8x} + x, \\
\phi'(x) &= \frac{(6x^2 + 8x)(3x^2 + 8x) - (6x + 8)(2x^3 + 4x^2 + 10)}{(3x^2 + 8x)^2} \\
&= \frac{(6x + 8)f(x)}{(3x^2 + 8x)^2},
\end{aligned}$$

and $\phi(\alpha) = \alpha$. We easily deduce that $\phi'(\alpha) = 0$, since $f(\alpha) = 0$. Consequently, the proposed method converges (at least) quadratically.

Exercise 2.17 The proposed method is convergent at least with order 2 since $\phi'(\alpha) = 0$.

Exercise 2.18 By keeping the remaining parameters unchanged, the method converges after 30 iterations to the value 0.641182210863894 which differs by less than 10^{-7} from the result previously computed in Solution 2.11. However, the behavior of the function, which is quite flat near $x = 0$, suggests that the result computed previously could be more accurate. In Figure 10.2, right, we show the graph of f in $(0.5, 0.7)$, obtained by the following instructions:

```
f=@(x) x^3-3*x^2*2^(-x)+3*x*4^(-x)-8^(-x);
fplot(f,[0.5 0.7]);
grid on
```

10.3 Chapter 3

Exercise 3.1 Since $x \in (x_0, x_n)$, there exists an interval $I_i = (x_{i-1}, x_i)$ such that $x \in I_i$. We can easily see that $\max_{x \in I_i} |(x - x_{i-1})(x - x_i)| = h^2/4$. If we bound $|x - x_{i+1}|$ above by $2h$, $|x - x_{i-2}|$ by $3h$ and so on, we obtain the inequality (3.6).

Exercise 3.2 In all cases we have $n = 4$ and thus we should estimate the fifth derivative of each function in the given interval. We find: $\max_{x\in[-1,1]} |f_1^{(5)}| \simeq 1.18$, $\max_{x\in[-1,1]} |f_2^{(5)}| \simeq 1.54$, $\max_{x\in[-\pi/2,\pi/2]} |f_3^{(5)}| \simeq 1.41$. Thanks to formula (3.7), the upper bounds for the corresponding errors are about 0.0018, 0.0024 and 0.0211, respectively.

Exercise 3.3 Using the MATLAB command `polyfit` we compute the interpolating polynomials of degree 3 in the two cases:

```
year=[1975 1980 1985 1990];
west=[72.8 74.2 75.2 76.4];
east=[70.2 70.2 70.3 71.2];
cwest=polyfit(year,west,3);
ceast=polyfit(year,east,3);
estwest=polyval(cwest,[1977 1983 1988]);
esteast=polyval(ceast,[1977 1983 1988]);
```

The estimated values in 1977, 1983 and 1988 are

```
estwest =
   73.4464   74.8096   75.8576
esteast =
   70.2328   70.2032   70.6992
```

for the Western and Eastern Europe, respectively.

Exercise 3.4 We choose the month as time-unit. The initial time $t_0 = 1$ corresponds to November 1987, while $t_7 = 157$ to November 2000. With the following instructions we compute the coefficients of the polynomial interpolating the given prices:

```
time = [1 14 37 63 87 99 109 157];
price = [4.5 5 6 6.5 7 7.5 8 8];
[c] = polyfit(time,price,7);
```

Setting `[price2002]=polyval(c,181)` we find that the estimated price of the magazine in November 2002 is approximately 11.24 euros.

Exercise 3.5 In this special case, since the number of interpolation nodes is 4, the interpolatory cubic spline, computed by the command `spline`, coincides with the interpolating polynomial. As a matter of fact, the spline interpolates the nodal data, moreover its first and second derivatives are continuous while the third derivative is continuous at the internal nodes x_1 and x_2, thanks to the *not-a-knot* condition used by MATLAB. This wouldn't be true for the natural interpolating cubic spline.

Exercise 3.6 We use the following instructions:

```
T = [4:4:20];
rho=[1000.7794,1000.6427,1000.2805,999.7165,998.9700];
Tnew = [6:4:18]; format long e;
rhonew = spline(T,rho,Tnew)

rhonew =
  Columns 1 through 2
     1.000740787500000e+03     1.000488237500000e+03
  Columns 3 through 4
     1.000022450000000e+03     9.993649250000000e+02
```

The comparison with the further measures shows that the approximation is extremely accurate. Note that the state equation for the sea-water (UNESCO, 1980) assumes a fourth-order dependence of the density on the temperature. However, the coefficient of the fourth power of T is of the order of 10^{-9} and the cubic spline provides a good approximation of the measured values.

Exercise 3.7 We compare the results computed using the interpolatory cubic spline obtained using the MATLAB command `spline` (denoted with `s3`), the interpolatory natural spline (`s3n`) and the interpolatory spline with null first derivatives at the endpoints of the interpolatory interval (`s3d`) (computed with Program 3.2). We use the following instructions:

```
year=[1965 1970 1980 1985 1990 1991];
production=[17769 24001 25961 34336 29036 33417];
z=[1962:0.1:1992];
s3  = spline(year,production,z);
s3n = cubicspline(year,production,z);
s3d = cubicspline(year,production,z,0,[0 0]);
```

In the following table we resume the computed values (expressed in thousands of tons of goods):

year	1962	1977	1992
`s3`	514.6	2264.2	4189.4
`s3n`	1328.5	2293.4	3779.8
`s3d`	2431.3	2312.6	2216.6

The comparison with the real data (1238, 2740.3 and 3205.9 thousands of tons, respectively) shows that the values predicted by the natural spline are accurate also outside the interpolation interval (see Figure 10.3, left). On the contrary, the interpolating polynomial introduces large oscillations near this end-point and underestimates the production of as many as -7768.5×10^6 Kg for 1962.

Exercise 3.8 The interpolating polynomial `p` and the spline `s3` can be evaluated by the following instructions:

```
pert = 1.e-04;
x=[-1:2/20:1]; y=sin(2*pi*x)+(-1).^[1:21]*pert;
z=[-1:0.01:1]; c=polyfit(x,y,20);
p=polyval(c,z); s3=spline(x,y,z);
```

When we use the unperturbed data (`pert=0`) the graphs of both `p` and `s3` are indistinguishable from that of the given function. The situation changes dramatically when the perturbed data are used (`pert=1.e-04`). In particular, the interpolating polynomial shows strong oscillations at the end-points of the interval, whereas the spline remains practically unchanged (see Figure 10.3, right). This example shows that approximation by splines is in general more stable with respect to perturbation errors than the global Lagrange interpolation.

Exercise 3.9 If $n = m$, setting $\tilde{f} = \Pi_n f$ we find that the first member of (3.28) is null. Thus in this case $\Pi_n f$ is the solution of the least-squares problem. Since the interpolating polynomial is unique, we deduce that this is the unique solution to the least-squares problem.

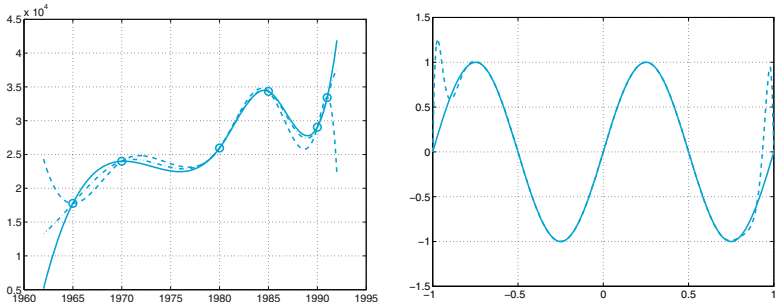

Figure 10.3. At left: comparison among the cubic spline for the data of Exercise 3.7: `s3` (*solid line*), `s3d` (*dashed line*) and `s3n` (*dotted line*). The circles denote the values used in the interpolation. At right: the interpolating polynomial (*dashed line*) and the interpolatory cubic spline (*solid line*) corresponding to the perturbed data (Solution 3.8). Note the severe oscillations of the interpolating polynomial near the end-points of the interval

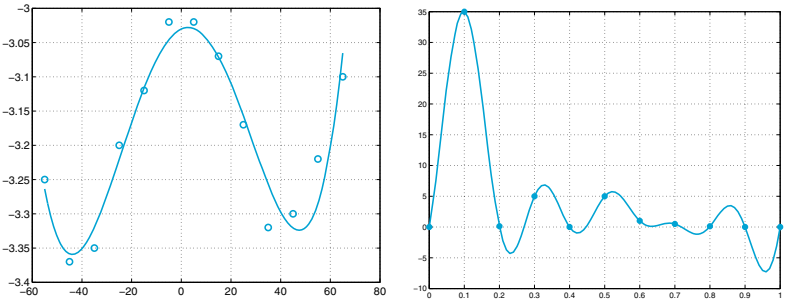

Figure 10.4. At left: least-squares polynomial of degree 4 (*solid line*) compared with the data in the first column of Table 3.1. (Solution 3.10). At right: the trigonometric interpolant obtained using the instructions in Solution 3.14. Circles refer to the available experimental data

Exercise 3.10 The coefficients (obtained by the command `polyfit`) of the requested polynomials are (only the first 4 significant digits are shown):

$K = 0.67$, $a_4 = 7.211\ 10^{-8}$, $a_3 = -6.088\ 10^{-7}$, $a_2 = -2.988\ 10^{-4}$, $a_1 = 1.650\ 10^{-3}$, $a_0 = -3.030$;
$K = 1.5$, $a_4 = -6.492\ 10^{-8}$, $a_3 = -7.559\ 10^{-7}$, $a_2 = 3.788\ 10^{-4}$, $a_1 = 1.67310^{-3}$, $a_0 = 3.149$;
$K = 2$, $a_4 = -1.050\ 10^{-7}$, $a_3 = 7.130\ 10^{-8}$, $a_2 = 7.044\ 10^{-4}$, $a_1 = -3.828\ 10^{-4}$, $a_0 = 4.926$;
$K = 3$, $a_4 = -2.319\ 10^{-7}$, $a_3 = 7.740\ 10^{-7}$, $a_2 = 1.419\ 10^{-3}$, $a_1 = -2.574\ 10^{-3}$, $a_0 = 7.315$.

In Figure 10.4, left, we show the graph of the polynomial computed using the data in the column with $K = 0.67$ of Table 3.1.

Exercise 3.11 By repeating the first 3 instructions reported in Solution 3.7 and using the command `polyfit`, we find the following values (in 10^5 Kg): 15280.12 in 1962; 27407.10 in 1977; 32019.01 in 1992, which represent good approximations to the real ones (12380, 27403 and 32059, respectively).

Exercise 3.12 We can rewrite the coefficients of the system (3.30) in terms of mean and variance by noting that the variance can be expressed as $v = \frac{1}{n+1}\sum_{i=0}^{n} x_i^2 - M^2$. Thus the coefficients of the first equation are $(n+1)$ and M, while those of the second equation are M and $(n+1)(v+M^2)$.

Exercise 3.13 The equation of the least-squares straight line is $y = a_0 + a_1 x$, where a_0 and a_1 are the solutions of system (3.30). The first equation of (3.30) provides that the point, whose abscissa is M and ordinate is $\sum_{i=0}^{n} y_i/(n+1)$, belongs to the least-squares straight line.

Exercise 3.14 We can use the command `interpft` as follows:

```
discharge = [0 35 0.125 5 0 5 1 0.5 0.125 0];
y =interpft(discharge,100);
```

The graph of the obtained solution is reported in Figure 10.4, right.

10.4 Chapter 4

Exercise 4.1 Using the following second-order Taylor expansions of f at the point x_0, we obtain

$$f(x_1) = f(x_0) + hf'(x_0) + \tfrac{h^2}{2}f''(x_0) + \tfrac{h^3}{6}f'''(\xi_1),$$
$$f(x_2) = f(x_0) + 2hf'(x_0) + 2h^2 f''(x_0) + \tfrac{4h^3}{3}f'''(\xi_2),$$

where $\xi_1 \in (x_0, x_1)$ and $\xi_2 \in (x_0, x_2)$. Replacing these two expressions in the first formula of (4.11), yields

$$\frac{1}{2h}[-3f(x_0) + 4f(x_1) - f(x_2)] = f'(x_0) + \frac{h^2}{3}[f'''(\xi_1) - 2f'''(\xi_2)],$$

then the thesis follows for a suitable $\xi_0 \in (x_0, x_2)$. A similar procedure can be used for the formula at x_n.

Exercise 4.2 By writing the second-order Taylor expansions of $f(\overline{x} \pm h)$ around $\overline{x}$, we have

$$f(\bar{x} \pm h) = f(\bar{x}) \pm hf'(\bar{x}) + \frac{h^2}{2}f''(\bar{x}) \pm \frac{h^3}{6}f'''(\xi_\pm),$$

with $\xi_- \in (\bar{x}-h, \bar{x})$ and $\xi_+ \in (\bar{x}, \bar{x}+h)$. Subtracting these two expressions and dividing by $2h$ we obtain formula (4.10) which is a second-order approximation of $f'(\overline{x})$.

Exercise 4.3 Assuming that $f \in C^4$ and proceeding as in Solution 4.2 we obtain the following errors:

$$a. \quad -\frac{1}{4}f^{(4)}(\xi)h^3, \quad b. \quad -\frac{1}{12}f^{(4)}(\xi)h^3, \quad c. \quad \frac{1}{6}f^{(4)}(\xi)h^3.$$

Exercise 4.4 Using the approximation (4.9), we obtain the following values:

t (months)	0	0.5	1	1.5	2	2.5	3
δn	–	78	45	19	7	3	–
n'	–	77.91	39.16	15.36	5.91	1.99	–

By comparison with the exact values of $n'(t)$ we can conclude that the computed values are sufficiently accurate.

Exercise 4.5 The quadrature error can be bounded by

$$(b-a)^3/(24M^2) \max_{x\in[a,b]} |f''(x)|,$$

where $[a, b]$ is the integration interval and M the (unknown) number of subintervals.

The function f_1 is infinitely differentiable. From the graph of f_1'' we infer that $|f_1''(x)| \leq 2$ in the integration interval. Thus the integration error for f_1 is less than 10^{-4} provided that $2 \cdot 5^3/(24M^2) < 10^{-4}$, that is $M > 322$.

Also the function f_2 is differentiable to any order. Since $\max_{x\in[0,\pi]} |f_2''(x)| = \sqrt{2}e^{3\pi/4}$, the integration error is less than 10^{-4} provided that $M > 439$. These inequalities actually provide an over estimation of the integration errors. Indeed, the (effective) minimum number of intervals which ensures that the error is below the fixed tolerance of 10^{-4} is much lower than that predicted by our result (for instance, for the function f_1 this number is 71). Finally, we note that, since f_3 is not differentiable at both $x = 0$ and $x = 1$, the theoretical error estimate doesn't hold.

Exercise 4.6 On each interval I_k, $k = 1, \ldots, M$, the error is equal to $H^3/24 f''(\xi_k)$ with $\xi_k \in [x_{k-1}, x_k]$ and hence the global error will be $H^3/24$ $\sum_{k=1}^{M} f''(\xi_k)$. Since f'' is a continuous function in $[a, b]$ there exists a point $\xi \in [a, b]$ such that $f''(\xi) = \frac{1}{M}\sum_{k=1}^{M} f''(\xi_k)$. Using this result and the fact that $MH = b - a$, we derive equation (4.14).

Exercise 4.7 This effect is due to the accumulation of local errors on each sub-interval.

Exercise 4.8 By construction, the mid-point formula integrates exactly the constants. To verify that the linear polynomials also are exactly integrated, it is sufficient to verify that $I(x) = I_{PM}(x)$. As a matter of fact we have

$$I(x) = \int_a^b x \, dx = \frac{b^2 - a^2}{2}, \quad I_{PM}(x) = (b-a)\frac{b+a}{2}.$$

Exercise 4.9 For the function f_1 we find $M = 71$ if we use the trapezoidal formula and only $M = 8$ for the composite Gauss-Legendre formula with $n = 1$. (For this formula we can use Program 10.1.) Indeed, the computational advantage of this latter formula is evident.

Program 10.1. gausslegendre: Gauss-Legendre composite quadrature formula, with $n = 1$

```
function intGL=gausslegendre(a,b,f,M,varargin)
y = [-1/sqrt(3),1/sqrt(3)];
H2 = (b-a)/(2*M);
z = [a:2*H2:b];
zM = (z(1:end-1)+z(2:end))*0.5;
x = [zM+H2*y(1), zM+H2*y(2)];
f = f(x,varargin{:});
intGL = H2*sum(f);
return
```

Exercise 4.10 Equation (4.18) states that the quadrature error for the composite trapezoidal formula with $H = H_1$ is equal to CH_1^2, with $C = -\dfrac{b-a}{12} f''(\xi)$. If f'' does not vary "too much", we can assume that also the error with $H = H_2$ behaves like CH_2^2. Then, by equating the two expressions

$$I(f) \simeq I_1 + CH_1^2, \quad I(f) \simeq I_2 + CH_2^2, \tag{10.3}$$

we obtain $C = (I_1 - I_2)/(H_2^2 - H_1^2)$. Using this value in one of the expressions (10.3), we obtain equation (4.35), that is, a better approximation than the one produced by I_1 or I_2.

Exercise 4.11 We seek the maximum positive integer p such that $I_{appr}(x^p) = I(x^p)$. For $p = 0, 1, 2, 3$ we find the following nonlinear system with 4 equations in the 4 unknowns α, β, $\bar{x}$ and $\bar{z}$:

$$\begin{aligned}
p = 0 &\rightarrow \alpha + \beta = b - a, \\
p = 1 &\rightarrow \alpha\bar{x} + \beta\bar{z} = \frac{b^2 - a^2}{2}, \\
p = 2 &\rightarrow \alpha\bar{x}^2 + \beta\bar{z}^2 = \frac{b^3 - a^3}{3}, \\
p = 3 &\rightarrow \alpha\bar{x}^3 + \beta\bar{z}^3 = \frac{b^4 - a^4}{4}.
\end{aligned}$$

From the first two equations we can eliminate α and $\bar{z}$ and reduce the system to a new one in the unknowns β and $\bar{x}$. In particular, we find a second-order equation in β from which we can compute β as a function of $\bar{x}$. Finally, the nonlinear equation in $\bar{x}$ can be solved by the Newton method, yielding two values of $\bar{x}$ that are the nodes of the Gauss-Legendre quadrature formula with $n = 1$.

Exercise 4.12 Since

$$f_1^{(4)}(x) = 24\frac{1 - 10(x-\pi)^2 + 5(x-\pi)^4}{(1+(x-\pi)^2)^5},$$
$$f_2^{(4)}(x) = -4e^x \cos(x),$$

we find that the maximum of $|f_1^{(4)}(x)|$ is bounded by $M_1 \simeq 23$, while that of $|f_2^{(4)}(x)|$ by $M_2 \simeq 18$. Consequently, from (4.22) we obtain $H < 0.21$ in the first case and $H < 0.16$ in the second case.

Exercise 4.13 The MATLAB commands:

```
syms x
I=int(exp(-x^2/2),0,2);
Iex=eval(I)
```

yields the value 1.19628801332261 for the integral at hand.

The Gauss-Legendre formula applied to the same interval with $M = 1$ would provide the value 1.20278027622354 (with an absolute error equal to 6.4923e-03), while the simple Simpson formula gives 1.18715264069572 with a slightly larger error, equal to 9.1354e-03.

Exercise 4.14 We note that $I_k > 0\ \forall k$, since the integrand is non-negative. Therefore, we expect that all the values produced by the recursive formula should be non-negative. Unfortunately, the recursive formula is unstable to the propagation of roundoff errors and produces negative elements:

```
I(1)=1/exp(1); for k=2:20, I(k)=1-k*I(k-1); end
```

The result is `I(20) = 104.86` in MATLAB, while Octave produces `I(20) = -30.1924`. Using the composite Simpson formula, with $M \geq 16$, we can compute the integral with the desired accuracy, as a matter of fact, denoting by $f(x)$ the integrand function, the absolute value of its fourth derivative is bounded by $M \simeq 1.46\ 10^5$. Consequently, from (4.22) we obtain $H < 0.066$.

Exercise 4.15 The idea of Richardson's extrapolation is general and can be applied to any quadrature formula. By proceeding as in Solution 4.10, recalling that both Simpson and Gauss quadrature formulas are fourth-order accurate, formula (4.35) reads

$$I_R = I_1 + (I_1 - I_2)/(H_2^4/H_1^4 - 1).$$

For the Simpson formula we obtain

$$I_1 = 1.19616568040561,\ I_2 = 1.19628173356793,\ \Rightarrow\ I_R = 1.19628947044542,$$

with an absolute error $I(f) - I_R = -1.4571e-06$ (we gain two orders of magnitude with respect to I_1 and a factor 1/4 with respect to I_2). Using the Gauss-Legendre formula we obtain (the errors are reported between parentheses):

$$I_1 = 1.19637085545393\ \ (-8.2842e-05),$$
$$I_2 = 1.19629221796844\ \ (-4.2046e-06),$$
$$I_R = 1.19628697546941\ \ (1.0379e-06).$$

The advantage of using the Richardson extrapolation method is evident.

Exercise 4.16 We must compute by the Simpson formula the values $j(r,0) = \sigma/(\varepsilon_0 r^2)\int_0^r f(\xi)d\xi$ with $r = k/10$, for $k = 1,\ldots,10$ and $f(\xi) = e^\xi\xi^2$.

In order to estimate the integration error we need the fourth derivative $f^{(4)}(\xi) = e^\xi(\xi^2 + 8\xi + 12)$. The maximum of $f^{(4)}$ in the integration interval $[0,r]$ is attained at $\xi = r$ since $f^{(4)}$ is monotonically increasing. For a given r the error is below 10^{-10} provided that $H^4 < 10^{-10}2880/(rf^{(4)}(r))$. For $r = k/10$ with $k = 1,\ldots,10$ by the following instructions we can compute the minimum numbers of subintervals which ensure that the previous inequalities are satisfied:

```
r=[0.1:0.1:1]; maxf4=exp(r).*(r.^2+8*r+12);
H=(10^(-10)*2880./(r.*maxf4)).^(1/4); M=fix(r./H)

M =
     4    11    20    30    41    53    67    83   100
118
```

Therefore, the values of $j(r,0)$ are computed by running the following instructions:

```
sigma=0.36; epsilon0 = 8.859e-12;
f=@(x) exp(x).*x.^2;
for k = 1:10
   r = k/10;
   j(k)=simpsonc(0,r,M(k),f);
   j(k) = j(k)*sigma/(r^2*epsilon0);
end
```

Exercise 4.17 We compute $E(213)$ using the Simpson composite formula by increasing the number of intervals until the difference between two consecutive approximations (divided by the last computed value) is less than 10^{-11}:

```
f=@(x) 1./(x.^5.*(exp(1.432./(213*x))-1));
a=3.e-04; b=14.e-04;
i=1; err = 1; Iold = 0; while err >= 1.e-11
I=2.39e-11*simpsonc(a,b,i,f);
err = abs(I-Iold)/abs(I);
Iold=I;
i=i+1;
end
```

The procedure returns the value $i = 59$. Therefore, using 58 equispaced intervals we can compute the integral $E(213)$ with ten exact significant digits. The same result could be obtained by the Gauss-Legendre formula using 53 intervals. Note that as many as 1609 intervals would be nedeed if using the composite trapezoidal formula.

Exercise 4.18 On the whole interval the given function is not regular enough to allow the application of the theoretical convergence result (4.22). One possibility is to decompose the integral into the sum of two intervals, $[0, 0.5]$ and $[0.5, 1]$, in which the function is regular (it is actually a polynomial of degree 2 in each sub-interval). In particular, if we use the Simpson rule on each interval we can even integrate f exactly.

10.5 Chapter 5

Exercise 5.1 Let x_n denote the number of algebraic operations (sums, subtractions and multiplications) required to compute one determinant of a matrix of order $n \times n$ by the Laplace rule (1.8). The following recursive formula holds

$$x_k - kx_{k-1} = 2k - 1, \qquad k \geq 2,$$

with $x_1 = 0$. Multiplying both sides of this equation by $1/k!$, we obtain

$$\frac{x_k}{k!} - \frac{x_{k-1}}{(k-1)!} = \frac{2k-1}{k!}$$

and summing both sides from 2 to n gives the solution:

$$x_n = n! \sum_{k=2}^{n} \frac{2k-1}{k!}.$$

Recalling that $\displaystyle\sum_{k=0}^{\infty} \frac{1}{k!} = e$, it holds

$$\sum_{k=2}^{n} \frac{2k-1}{k!} = 2\sum_{k=1}^{n-1} \frac{1}{k!} - \sum_{k=2}^{n} \frac{1}{k!} \simeq 2.718,$$

whence $x_n \simeq 3n!$. It is worth mentioning that the Cramer rule (see Section 5.2) requires about $3(n+1)!$ operations to solve a square linear system of order n with full matrix.

Exercise 5.2 We use the following MATLAB commands to compute the determinants and the corresponding CPU-times:

```
t = []; NN=3:500;
for n = NN
A=magic(n); tt=cputime; d=det(A); t=[t, cputime-tt];
end
```

Let us compute the coefficients of the cubic least-squares polynomial that approximate the data `NN=[3:500]` and `t`

```
c=polyfit(NN,t,3)
c =
   1.4055e-10   7.1570e-08  -3.6686e-06   3.1897e-04
```

If we compute the fourth degree least-squares polynomial

```
c=polyfit(NN,t,4)
```

we obtain the following coefficients:

```
c =
  7.6406e-15   1.3286e-10   7.4064e-08  -3.9505e-06   3.2637e-04
```

that is, the coefficient of n^4 is close to the machine precision while the other ones are quite unchanged with respect to the projection on $\mathbb{P}_3$. From this result, we can conclude that in MATLAB the CPU-time required for computing the determinant of a matrix of dimension `n` scales as $\mathtt{n}^3$.

Exercise 5.3 Denoting by A_i the principal submatrix of A of order i, we have: $\det A_1 = 1$, $\det A_2 = \varepsilon$, $\det A_3 = \det A = 2\varepsilon + 12$. Consequently, if $\varepsilon = 0$ the second principal submatrix is singular and the LU factorization of A does not exist (see Proposition 5.1). The matrix A is singular if $\varepsilon = -6$. In this case the LU factorization exists and yields

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1.25 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 7 & 3 \\ 0 & -12 & -4 \\ 0 & 0 & 0 \end{bmatrix}.$$

Nevertheless, note that U is singular (as we could have predicted since A is singular) and the upper triangular system $U\mathbf{x} = \mathbf{y}$ admits infinite solutions. We notice that the backward substitutions (5.10) cannot be applied because of the same reason.

Exercise 5.4 Let us consider algorithm 5.13. At step $k = 1$, $n - 1$ divisions were used to calculate the l_{i1} entries for $i = 2, \ldots, n$. Then $(n-1)^2$ multiplications and $(n-1)^2$ additions were used to create the new entries $a_{ij}^{(2)}$, for $i, j = 2, \ldots, n$. At step $k = 2$, the number of divisions is $(n-2)$, while the number of multiplications and additions will be $(n-2)^2$. At final step $k = n-1$ only 1 addition, 1 multiplication and 1 division is required. Thus, using the identies

$$\sum_{s=1}^{q} s = \frac{q(q+1)}{2}, \qquad \sum_{s=1}^{q} s^2 = \frac{q(q+1)(2q+1)}{6}, \quad q \geq 1,$$

we can conclude that to complete the LU factorization we need the following number of operations

$$\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 2 \right) = \sum_{k=1}^{n-1} (n-k)(1 + 2(n-k))$$
$$= \sum_{j=1}^{n-1} j + 2 \sum_{j=1}^{n-1} j^2 = \frac{(n-1)n}{2} + 2\frac{(n-1)n(2n-1)}{6} = \frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}.$$

Exercise 5.5 By definition, the inverse X of a matrix $A \in \mathbb{R}^{n \times n}$ satisfies $XA = AX = I$. Therefore, for $j = 1, \ldots, n$ the column vector $\mathbf{x}_j$ of X is the solution of the linear system $A\mathbf{x}_j = \mathbf{e}_j$, where $\mathbf{e}_j$ is the jth vector of the canonical basis of $\mathbb{R}^n$ with all components equal to zero except the jth that is equal to 1. After computing the LU factorization of A, the computation of the inverse of A requires the solution of n linear systems with the same matrix and different right-hand sides.

Exercise 5.6 Using the Program 5.1 we compute the L and U factors:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & -3.38 \cdot 10^{15} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 & 3 \\ 0 & -8.88 \cdot 10^{-16} & 14 \\ 0 & 0 & 4.73 \cdot 10^{-16} \end{bmatrix}.$$

If we compute their product we obtain the matrix

```
L*U
ans =
    1.0000    1.0000    3.0000
    2.0000    2.0000   20.0000
    3.0000    6.0000    0.0000
```

which differs from A since the entry in position (3,3) is equal to 0 while in A it is equal to 4.
The accurate computation of both L and U can be accomplished by invoking a partial pivoting by rows, indeed by the instruction `[L,U,P]=lu(A)` we obtain the correct results.

Exercise 5.7 Usually, only the triangular (upper or lower) part of a symmetric matrix is stored. Therefore, any operation that does not respect the symmetry of the matrix is not optimal in view of the memory storage. This is the case when row pivoting is carried out. A possibility is to exchange simultaneously rows and columns having the same index, limiting therefore the choice of the pivot only to the diagonal elements. More generally, a pivoting strategy involving exchange of rows and columns is called *complete pivoting* (see, e.g., [QSS07, Chap. 3]).

Exercise 5.8 The symbolic computation of the L and U factors yields

$$\mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ (\varepsilon-2)/2 & 1 & 0 \\ 0 & -1/\varepsilon & 1 \end{bmatrix}, \mathrm{U} = \begin{bmatrix} 2 & -2 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & 3 \end{bmatrix},$$

thus $l_{32} \to \infty$, when $\varepsilon \to 0$. If we choose $\mathbf{b} = (0, \varepsilon, 2)^T$, it is easy to verify that $\mathbf{x} = (1, 1, 1)^T$ is the exact solution of $\mathrm{A}\mathbf{x} = \mathbf{b}$. To analyze the error with respect to the exact solution for $\varepsilon \to 0$, let us take $\varepsilon = 10^{-k}$, for $k = 0, \ldots, 9$. The following instructions

```
e=1;xex=ones(3,1);err=[];
for k=1:10
b=[0;e;2];
L=[1 0 0; (e-2)*0.5 1 0; 0 -1/e 1];
U=[2 -2 0; 0 e 0; 0 0 3];
y=L\b; x=U\y;
err(k)=norm(x-xex)/norm(xex); e=e*0.1;
end
```

yield

```
   err =
      0     0     0     0     0     0     0     0     0     0
```

i.e., the numerical solution is not affected by rounding errors. This can be explained by noticing that all the entries of L, U and $\mathbf{b}$ are floating point numbers not affected by rounding errors and, unusually, no rounding errors are propagated during both forward and backward substitutions, even if the condition number of A is proportional to $1/\varepsilon$.
On the contrary, by setting $\mathbf{b} = (2\log(2.5) - 2, (\varepsilon - 2)\log(2.5) + 2, 2)^T$, which corresponds to the exact solution $\mathbf{x} = (\log(2.5), 1, 1)^T$, and analyzing the relative error for $\varepsilon = 1/3 \cdot 10^{-k}$, for $k = 0, \ldots, 9$, the instructions

```
e=1/3; xex=[log(5/2),1,1]'; err=[];
for k=1:10
b=[2*log(5/2)-2,(e-2)*log(5/2)+2,2]';
L=[1 0 0; (e-2)*0.5 1 0; 0 -1/e 1];
U=[2 -2 0; 0 e 0; 0 0 3];
y=L\b; x=U\y;
err(k)=norm(x-xex)/norm(xex); e=e*0.1;
end
```

provide

```
err =
  Columns 1 through 5
   1.8635e-16   5.5327e-15   2.6995e-14   9.5058e-14   1.3408e-12
  Columns 6 through 10
   1.2828e-11   4.8726e-11   4.5719e-09   4.2624e-08   2.8673e-07
```

In the latter case the error depends on the condition number of A, which obeys the law $K(A) = C/\varepsilon$ and satisfies the estimate (5.34).

Exercise 5.9 The computed solutions become less and less accurate when i increases. Indeed, the error norms are equal to $1.10 \cdot 10^{-14}$ for $i = 1$, to $9.32 \cdot 10^{-10}$ for $i = 2$ and to $2.51 \cdot 10^{-7}$ for $i = 3$. (We warn the reader that these results indeed change depending upon the different MATLAB versions used!!) This can be explained by observing that the condition number of A_i increases as i increases. Indeed, using the command `cond` we find that the condition number of A_i is $\simeq 10^3$ for $i = 1$, $\simeq 10^7$ for $i = 2$ and $\simeq 10^{11}$ for $i = 3$.

Exercise 5.10 If $(\lambda, \mathbf{v})$ are an eigenvalue-eigenvector pair of a matrix A, then λ^2 is an eigenvalue of A^2 with the same eigenvector. Indeed, from $\mathrm{A}\mathbf{v} = \lambda\mathbf{v}$ follows $\mathrm{A}^2\mathbf{v} = \lambda\mathrm{A}\mathbf{v} = \lambda^2\mathbf{v}$. Consequently, if A is symmetric and positive definite $K(\mathrm{A}^2) = (K(\mathrm{A}))^2$.

Exercise 5.11 The iteration matrix of the Jacobi method is:

$$\mathrm{B}_J = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ -\alpha^{-1} & 0 & 0 \end{bmatrix}.$$

Its eigenvalues are $\{0, \alpha^{-1}, -\alpha^{-1}\}$. Thus the method converges if $|\alpha| > 1$.

The iteration matrix of the Gauss-Seidel method is

$$\mathrm{B}_{GS} = \begin{bmatrix} 0 & 0 & -\alpha^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & \alpha^{-2} \end{bmatrix}$$

with eigenvalues $\{0, 0, \alpha^{-2}\}$. Therefore, the method converges if $|\alpha| > 1$. In particular, since $\rho(\mathrm{B}_{GS}) = [\rho(\mathrm{B}_J)]^2$, the Gauss-Seidel converges more rapidly than the Jacobi method.

Exercise 5.12 A sufficient condition for the convergence of the Jacobi and the Gauss-Seidel methods is that A is strictly diagonally dominant. The second row of A satisfies the condition of diagonal dominance provided that $|\beta| < 5$. Note that if we require directly that the spectral radii of the iteration matrices are less than 1 (which is a sufficient and necessary condition for convergence), we find the (less restrictive) limitation $|\beta| < 25$ for both methods.

Exercise 5.13 The relaxation method in vector form is

$$(\mathrm{I} - \omega \mathrm{D}^{-1}\mathrm{E})\mathbf{x}^{(k+1)} = [(1-\omega)\mathrm{I} + \omega \mathrm{D}^{-1}\mathrm{F}]\mathbf{x}^{(k)} + \omega \mathrm{D}^{-1}\mathbf{b}$$

where A = D − (E + F), D being the diagonal of A, and -E and -F the lower (resp. upper) part of A. The corresponding iteration matrix is

$$\mathrm{B}(\omega) = (\mathrm{I} - \omega \mathrm{D}^{-1}\mathrm{E})^{-1}[(1-\omega)\mathrm{I} + \omega \mathrm{D}^{-1}\mathrm{F}].$$

If we denote by λ_i the eigenvalues of $\mathrm{B}(\omega)$, we obtain

$$\begin{aligned}\left|\prod_{i=1}^{n}\lambda_i\right| &= |\mathrm{detB}(\omega)| \\ &= |\det[(I - \omega D^{-1}E)^{-1}]| \cdot |\det[(1-\omega)I + \omega D^{-1}F)]|.\end{aligned}$$

Noticing that, given two matrices A and B with $\mathrm{A} = \mathrm{I} + \alpha\mathrm{B}$, for any $\alpha \in \mathbb{R}$ it holds $\lambda_i(\mathrm{A}) = 1 + \alpha\lambda_i(\mathrm{B})$, and that all the eigenvalues of both $\mathrm{D}^{-1}E$ and $\mathrm{D}^{-1}F$ are null, we have

$$\left|\prod_{i=1}^{n}\lambda_i\right| = \left|\prod_{i=1}^{n}\frac{(1-\omega) + \omega\lambda_i(\mathrm{D}^{-1}F)}{1 - \omega\lambda_i(\mathrm{D}^{-1}E)}\right| = |1-\omega|^n.$$

Therefore, at least one eigenvalue must satisfy the inequality $|\lambda_i| \geq |1-\omega|$. Thus, a necessary condition to ensure convergence is that $|1-\omega| < 1$, that is, $0 < \omega < 2$.

Exercise 5.14 Matrix $\mathrm{A} = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$ is strictly diagonally dominant by rows, a sufficient condition for the Gauss-Seidel method to converge. On the contrary, matrix $\mathrm{A} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ is not strictly diagonally dominant by rows, however it is symmetric. Moreover, we can easily verify that it is positive definite, i.e. $\mathbf{z}^T\mathrm{A}\mathbf{z} > 0$ for any $\mathbf{z} \neq \mathbf{0}$ of $\mathbb{R}^2$. We perform the following computations by MATLAB (obviously, for this simple case, we could perform them by hands!):

```
syms z1 z2 real
z=[z1;z2]; A=[1 1; 1 2];
pos=z'*A*z; simple(pos)
ans =
   z1^2+2*z1*z2+2*z2^2
ans =
   z1^2+2*z1*z2+2*z2^2
```

where the command `syms z1 z2 real` converts the variables `z1` and `z2` from symbolic to real type, while the command `simple` tries several algebraic simplifications of `pos` and returns the shortest. It is easy to see that the computed quantity is positive since it can be rewritten as `(z1+z2)^2+z2^2`. Thus, the given matrix is symmetric and positive definite, a sufficient condition for the Gauss-Seidel method to converge.

Exercise 5.15 We find:

for the Jacobi method: $\begin{cases} x_1^{(1)} = \frac{1}{2}(1 - x_2^{(0)}), \\ x_2^{(1)} = -\frac{1}{3}(x_1^{(0)}); \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{4}, \\ x_2^{(1)} = -\frac{1}{3}; \end{cases}$

for the Gauss-Seidel method: $\begin{cases} x_1^{(1)} = \frac{1}{2}(1 - x_2^{(0)}), \\ x_2^{(1)} = -\frac{1}{3}x_1^{(1)}, \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = \frac{1}{4}, \\ x_2^{(1)} = -\frac{1}{12}; \end{cases}$

for the gradient method, we first compute the initial residual

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathrm{A}\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \mathbf{x}^{(0)} = \begin{bmatrix} -3/2 \\ -5/2 \end{bmatrix}.$$

Then, since

$$\mathrm{P}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix},$$

we have $\mathbf{z}^{(0)} = \mathrm{P}^{-1}\mathbf{r}^{(0)} = (-3/4, -5/6)^T$. Therefore

$$\alpha_0 = \frac{(\mathbf{z}^{(0)})^T \mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T \mathrm{A}\mathbf{z}^{(0)}} = \frac{77}{107},$$

and

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{z}^{(0)} = (197/428, -32/321)^T.$$

Exercise 5.16 In the stationary case, the eigenvalues of the matrix $\mathrm{B}_\alpha = \mathrm{I} - \alpha \mathrm{P}^{-1}\mathrm{A}$ are $\mu_i(\alpha) = 1 - \alpha\lambda_i$, λ_i being the ith eigenvalue of $\mathrm{P}^{-1}\mathrm{A}$. Then

$$\rho(\mathrm{B}_\alpha) = \max_{i=1,\ldots,n} |1 - \alpha\lambda_i| = \max\{|1 - \alpha\lambda_{min}|, |1 - \alpha\lambda_{max}|\}.$$

Thus, the optimal value of α (that is the value that minimizes the spectral radius of the iteration matrix) is the root of the equation

$$1 - \alpha\lambda_{min} = \alpha\lambda_{max} - 1$$

which yields (5.58). Formula (5.72) follows now by a direct computation of $\rho(\mathrm{B}_{\alpha_{opt}})$.

Exercise 5.17 We have to minimize the function $\Phi(\alpha) = \|\mathbf{e}^{(k+1)}\|_{\mathrm{A}}^2$ with respect to $\alpha \in \mathbb{R}$. Since $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)} = \mathbf{e}^{(k)} - \alpha\mathbf{z}^{(k)}$, we obtain

$$\Phi(\alpha) = \|\mathbf{e}^{(k+1)}\|_{\mathrm{A}}^2 = \|\mathbf{e}^{(k)}\|_{\mathrm{A}}^2 + \alpha^2\|\mathbf{z}^{(k)}\|_{\mathrm{A}}^2 - 2\alpha(\mathbf{z}^{(k)})^T\mathrm{A}\mathbf{e}^{(k)}.$$

The minimum of $\Phi(\alpha)$ is found in correspondence to the value α_k such that $\Phi'(\alpha_k) = 0$, i.e.,

$$\alpha_k\|\mathbf{z}^{(k)}\|_{\mathrm{A}}^2 - (\mathbf{z}^{(k)})^T\mathrm{A}\mathbf{e}^{(k)} = 0,$$

so that $\alpha_k = ((\mathbf{z}^{(k)})^T\mathrm{A}\mathbf{e}^{(k)})/\|\mathbf{z}^{(k)}\|_{\mathrm{A}}^2$. Finally, (5.60) follows by noticing that $\mathrm{A}\mathbf{e}^{(k)} = \mathbf{r}^{(k)}$.

Exercise 5.18 We provide two possible proofs.
1. Note that $\mathrm{P}^{-1}\mathrm{A} = \mathrm{P}^{-1/2}(\mathrm{P}^{-1/2}\mathrm{A}\mathrm{P}^{-1/2})\mathrm{P}^{1/2}$ where $\mathrm{P}^{1/2}$ is the square root of P (see, e.g. [QV94, Sect. 2.5]). Since P is symmetric positive definite, $\mathrm{P}^{1/2}$ is symmetric and positive definite and it is the unique solution of the matrix equation $\mathrm{X}^2 = \mathrm{P}$. This shows that $\mathrm{P}^{-1}\mathrm{A}$ is similar to the matrix $\mathrm{P}^{-1/2}\mathrm{A}\mathrm{P}^{-1/2}$ which is symmetric positive definite.
2. The eigenpairs $(\mu, \mathbf{y})$ of $\mathrm{P}^{-1}\mathrm{A}$ satisfy the equation $\mathrm{P}^{-1}\mathrm{A}\mathbf{y} = \mu\mathbf{y}$, that is $\mathrm{A}\mathbf{y} = \mu\mathrm{P}\mathbf{y}$, therefore $\mu = (\mathbf{y}^T A\mathbf{y})/(\mathbf{y}^T\mathrm{P}\mathbf{y}) > 0$ since both A and P are symmetric positive definite.

Exercise 5.19 The matrix associated to the Leontieff model is symmetric, but not positive definite. Indeed, using the following instructions:

```
for i=1:20;
  for j=1:20;
    C(i,j)=i+j;
  end;
end;
A=eye(20)-C;
[min(eig(A)), max(eig(A))]

ans =
 -448.58   30.583
```

we can see that the minimum eigenvalue is a negative number and the maximum eigenvalue is a positive number. Therefore, the convergence of the gradient method is not guaranteed. However, since A is nonsingular, the given system is equivalent to the system $\mathrm{A}^T\mathrm{A}\mathbf{x} = \mathrm{A}^T\mathbf{b}$, where $\mathrm{A}^T\mathrm{A}$ is symmetric and positive definite. We solve the latter by the gradient method requiring that the norm of the residual be less than 10^{-10} and starting from the initial data $\mathbf{x}^{(0)} = \mathbf{0}$:

```
b = [1:20]';  AA=A'*A; b=A'*b; x0 = zeros(20,1);
[x,iter]=itermeth(AA,b,x0,100,1.e-10);
```

The method converges in 15 iterations. A drawback of this approach is that the condition number of the matrix $\mathrm{A}^T\mathrm{A}$ is, in general, larger than the condition number of A.

10.6 Chapter 6

Exercise 6.1 A_1: the power method converges in 34 iterations to the value 2.00000000004989. A_2: starting from the same initial vector, the power method requires now 457 iterations to converge to the value 1.99999999990611. The slower convergence rate can be explained by observing that the two largest eigenvalues are very close one another. Finally, for the matrix A_3 the method doesn't converge since A_3 features two distinct eigenvalues (i and $-i$) of maximum modulus.

Exercise 6.2 The Leslie matrix associated with the values in the table is

$$A = \begin{bmatrix} 0 & 0.5 & 0.8 & 0.3 \\ 0.2 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0.8 & 0 \end{bmatrix}.$$

Using the power method we find $\lambda_1 \simeq 0.5353$. The normalized distribution of this population for different age intervals is given by the components of the corresponding unitary eigenvector, that is, $\mathbf{x}_1 \simeq (0.8477, 0.3167, 0.2367, 0.3537)^T$.

Exercise 6.3 We rewrite the initial guess as

$$\mathbf{y}^{(0)} = \beta^{(0)} \left(\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \sum_{i=3}^{n} \alpha_i \mathbf{x}_i \right),$$

with $\beta^{(0)} = 1/\|\mathbf{x}^{(0)}\|$. By calculations similar to those carried out in Section 6.2, at the generic step k we find:

$$\mathbf{y}^{(k)} = \gamma^k \beta^{(k)} \left(\alpha_1 \mathbf{x}_1 e^{ik\vartheta} + \alpha_2 \mathbf{x}_2 e^{-ik\vartheta} + \sum_{i=3}^{n} \alpha_i \frac{\lambda_i^k}{\gamma^k} \mathbf{x}_i \right).$$

Therefore, when $k \to \infty$, the first two terms don't vanish and, due to the opposite sign of the exponents, the sequence of the $\mathbf{y}^{(k)}$ oscillates and cannot converge.

Exercise 6.4 If A is non-singular, from the eigenvalue equation $A\mathbf{x} = \lambda\mathbf{x}$, we deduce $A^{-1}A\mathbf{x} = \lambda A^{-1}\mathbf{x}$, and therefore $A^{-1}\mathbf{x} = (1/\lambda)\mathbf{x}$.

Exercise 6.5 The power method applied to the matrix A generates an oscillating sequence of approximations of the maximum modulus eigenvalue (see, Figure 10.5). This behavior is due to the fact that the matrix A has two distinct eigenvalues of maximum modulus.

Exercise 6.6 Since the eigenvalues of a real symmetric matrix are all real, they lie inside a closed bounded interval $[\lambda_a, \lambda_b]$. Our aim is to estimate both λ_a and λ_b. To compute the eigenvalue of maximum modulus of A we use Program 6.1:

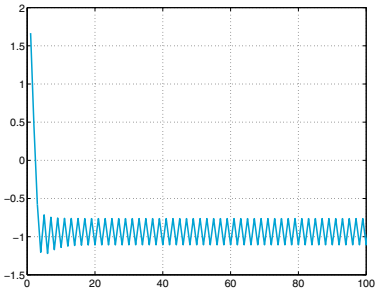

Figure 10.5. The approximations of the maximum modulus eigenvalue of the matrix of Solution 6.5 computed by the power method

```
A=wilkinson(7);
x0=ones(7,1); tol=1.e-15; nmax=100;
[lambdab,x,iter]=eigpower(A,tol,nmax,x0);
```

After 35 iterations we obtain `lambdab=3.76155718183189`. Since λ_a is the eigenvalue of A farest from λ_b, in order to compute it we apply the power method to the matrix $\mathrm{A}_b = \mathrm{A} - \lambda_b \mathrm{I}$, that is we compute the maximum modulus eigenvalue of the matrix A_b. Then we will set $\lambda_a = \lambda + \lambda_b$. The instructions

```
[lambda,x,iter]=eigpower(A-lambdab*eye(7),tol,nmax,x0);
lambdaa=lambda+lambdab
```

yield `lambdaa =-1.12488541976457` after 33 iterations. These results are satisfactory approximations of the extremal eigenvalues of A.

Exercise 6.7 Let us start by considering the matrix A. We observe that there is an isolated row circle centered at $(9, 0)$ with radius equal to 1, that can contain only one eigenvalue (say λ_1), in view of Proposition 6.1. Therefore $\lambda_1 \in \mathbb{R}$, more precisely $\lambda_1 \in (8, 10)$. Moreover, from Figure 10.6, right, we note that A features two other isolated column circles centered at $(2, 0)$ and $(4, 0)$, respectively, both with radius equal to $1/2$. Therefore A has two other real eigenvalues $\lambda_2 \in (1.5, 2.5)$ and $\lambda_3 \in (3.4, 4.5)$. Since all the coefficients of A are real, we can conclude that also the fourth eigenvalue will be real.

Let us consider now the matrix B that admits only one isolated column circle (see Figure 10.7 right), centered at $(-5, 0)$ and with radius $1/2$. Then, thanks to the previous consideration the corresponding eigenvalue must be real and it will belong to the interval $(-5.5, -4.5)$. The remaining eigenvalues can be either all real, or one real and 2 complex.

Exercise 6.8 The row circles of A feature an isolated circle of center (5,0) and radius 2 the maximum modulus eigenvalue must belong to. Therefore, we can set the value of the shift equal to 5. The comparison between the number of iterations and the computational cost of the power method with and without shift can be found using the following commands:

```
A=[5 0 1 -1; 0 2 0 -1/2; 0 1 -1 1; -1 -1 0 0];
```

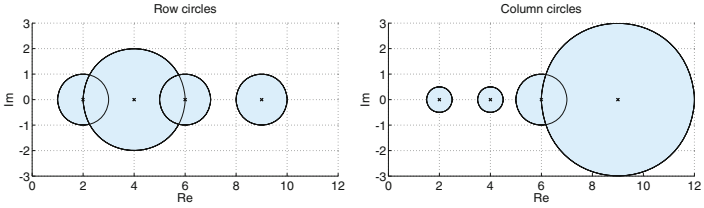

Figure 10.6. Row circles (at left) and column circles (at right) of the matrix A of Solution 6.7

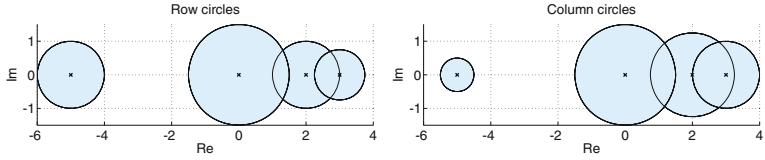

Figure 10.7. Row circles (at left) and column circles (at right) circles of the matrix B of Solution 6.7

```
tol=1e-14; x0=[1 2 3 4]';  nmax=1000;
tic; [lambda,x,iter]=eigpower(A,tol,nmax,x0);
toc, iter

Elapsed time is 0.001854 seconds.
iter = 35

tic; [lambda,x,iter]=invshift(A,5,tol,nmax,x0);
toc, iter

Elapsed time is 0.000865 seconds.
iter = 12
```

The power method with shift requires in this case a lower number of iterations (1 versus 3) and almost half the cost than the usual power method (also accounting for the extra time needed to compute the LU factorization of A off-line).

Exercise 6.9 It holds

$$\mathrm{A}^{(k)} = \mathrm{Q}^{(k+1)}\mathrm{R}^{(k+1)} \text{ and } \mathrm{A}^{(k+1)} = \mathrm{R}^{(k+1)}\mathrm{Q}^{(k+1)}$$

and then

$$(\mathrm{Q}^{(k+1)})^T\mathrm{A}^{(k)}\mathrm{Q}^{(k+1)} = \mathrm{R}^{(k+1)}\mathrm{Q}^{(k+1)} = \mathrm{A}^{(k+1)}.$$

Since $(\mathrm{Q}^{(k+1)})^T = (\mathrm{Q}^{(k+1)})^{-1}$ we can conclude that matrix $\mathrm{A}^{(k)}$ is similar to $\mathrm{A}^{(k+1)}$ for any $k \geq 0$.

Exercise 6.10 We can use the command `eig` in the following way: `[X,D]=eig(A)`, where `X` is the matrix whose columns are the unit eigenvectors of `A` and `D` is a diagonal matrix whose elements are the eigenvalues of `A`. For the matrices A and B of Exercise 6.7 we should execute the following instructions:

```
A=[2 -1/2 0 -1/2; 0 4 0 2; -1/2 0 6 1/2; 0 0 1 9];
sort(eig(A))
ans =
    2.0000
    4.0268
    5.8003
    9.1728
B=[-5 0 1/2 1/2; 1/2 2 1/2 0; 0 1 0 1/2; 0 1/4 1/2 3];
sort(eig(B))
ans =
   -4.9921
   -0.3038
    2.1666
    3.1292
```

The conclusions drawn on the basis of Proposition 6.1 are quite coarse.

10.7 Chapter 7

Exercise 7.1 By direct inspection on the plot of function f we find that there is a single minimizer in the interval $[-2, 1]$. We use the following instructions to call Program 7.7:

```
a=-2; b=1; tol=1.e-8; kmax=100;
[xmin,fmin,iter]=golden(f,a,b,tol,kmax)
```

Note that the tolerance for the stopping test is set to 10^{-8}. After 42 iterations we obtain `xmin=-3.660253989004456e-01` and `fmin=-1.194742596743503`. The method converges linearly (see (7.19)).
Using now the MATLAB command `fminbnd` with the instructions:

```
options=optimset('TolX',1.e-8);
[xminf,fminf,exitflag,output]=fminbnd(f,a,b,options)
```

the same problem is solved by the golden section method with quadratic interpolation. In this case convergence is achieved in 9 iterations to the point `xmin=-3.660254076197302e-01`.

Exercise 7.2 Given $\boldsymbol{\gamma}_i(t) = (x_i(t), y_i(t))$, for $i = 1, 2$, we need to minimize the distance

$$d(t) = \sqrt{(x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2}$$

or, equivalently, its square as function of t. To solve this one dimensional minimum problem we can use the golden section method with quadratic interpolation implemented in the function `fminbnd`. Using the following instructions

```
x1=@(t)7*cos(t/3+pi/2)+5; y1=@(t)-4*sin(t/3+pi/2)-3;
x2=@(t)6*cos(t/6-pi/3)-4; y2=@(t)-6*sin(t/6-pi/3)+5;
d=@(t)(x1(t)-x2(t))^2+(y1(t)-y2(t))^2;
ta=0; tb=20; options=optimset('TolX',1.e-8);
[tmin,dmin,exitflag,output]=fminbnd(d,ta,tb,options)
```

we converge after 10 iterations to the solution `tmin=8.438731484275010`. At that time, the two ships stand at minimal distance `dmin=5.691754805947144` nautical miles, eigth hours and a half after their departure.

Exercise 7.3 We define the cost function and represent it together with its contour lines on a circular domain centered at (-1,0) with radius 3 by the following instructions:

```
fun=@(x) x(1)^4+x(2)^4+x(1)^3+3*x(1)*...
         x(2)^2-3*x(1)^2-3*x(2)^2+10;
[r,theta]=meshgrid(0:.1:3,0:pi/25:2*pi);
x1=r.*cos(theta)-1; y1=r.*sin(theta);
[n,m]=size(x1);z1=zeros(n,m);
for i=1:n, for j=1:m
    z1(i,j)=fun([x1(i,j);y1(i,j)]);
end, end
figure(1); clf;  p1=mesh(x1,y1,z1);
set(p1,'Edgecolor',[0,1,1]); hold on
contour(x1,y1,z1,100,'Linecolor',[0.8,0.8,0.8]);
```

By a direct inspection we see that the cost function features a local maximizer, a saddle point and two global minimizers (being this function even with respect to the x_2 variable). Choosing $\mathbf{x}^{(0)} = (-3, 0)$ and setting a tolerance $\varepsilon = 10^{-8}$ for the stopping test, using the commands:

```
x0=[-3;0]; options=optimset('TolX',1.e-8);
[xm,fval,exitf,out]=fminsearch(fun,x0,options)
```

we find the minimizer `xm=[-2.1861e+00, 2.1861e+00]` after 181 iterations and having used 353 function evaluations. The second minimizer is therefore `xm=[-2.1861e+00, -2.1861e+00]` because of the parity property of the function.

We warn the reader that choosing `x0=[1;0]`, the `fminsearch` MATLAB function converges to the local maximizer $(.75000, .61237)$ instead than to the minimizer, whereas the `fminsearch` Octave function still converges to the minimizer $(-2.1861, 2.1861)$.

Exercise 7.4 Let us write the sequence $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$ as

$$x^{(k+1)} = x^{(0)} + \sum_{\ell=0}^{k} \alpha_\ell d^{(\ell)}.$$

Since $x^{(0)} = 3/2$ we find

$$\begin{aligned} x^{(k+1)} &= \frac{3}{2} + \left(2 + \frac{2}{3^{k+1}}\right)(-1)^{k+1} = \frac{3}{2} - 2\sum_{\ell=0}^{k}(-1)^{\ell} - \frac{1}{2} - \frac{1}{6}\left(-\frac{1}{3}\right)^{k} \\ &= (-1)^{k+1}\left(1 + \frac{1}{6 \cdot 3^{k}}\right). \end{aligned}$$

Note that $x^{(k)}$ does not converge to zero eventhough the sequence $f(x^{(k)})$ is decreasing, as can be seen from Figure 10.8, left. When the points $x^{(k)}$ are near to $+1$ and -1, the first Wolfe condition (7.43) is not fulfilled since the variation of f between two steps becomes infinitesimal while the steplength is about the same (*circa* 2).

Exercise 7.5 By proceeding as done in the previous Exercise, we find $x^{(0)} = -2$ and $x^{(k+1)} = -2 + (1 - 3^{-k})/2 \to -3/2$ when $k \to \infty$. Also in this case

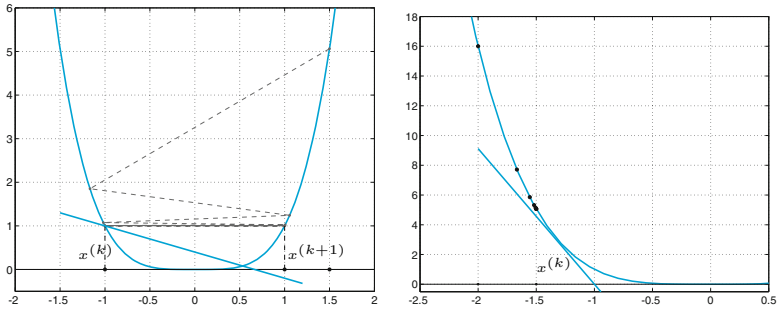

Figure 10.8. At left, the sequence yielded by the descent method of Exercise 7.4. Taking $x^{(k)} \simeq -1$, the point $(x^{(k+1)}, f(x^{(k+1)}))$ should stay beneath the blue straight line in order to satisfy the first Wolfe's condition with $\sigma = 0.2$; on the contrary it lies largely above, indeed $(x^{(k+1)}, f(x^{(k+1)})) \simeq (1, 1)$. At right, the sequence generated for Exercise 7.5. The point $(x^{(k+1)}, f(x^{(k+1)}))$ should stay at the right to the point where the blue straight line is tangent to the blue curve in order for the second Wolfe's condition with $\delta = 0.9$ to be satisifed; instead, it is close to $(-1.5, 5.06)$

the sequence of values $f(x^{(k)})$ is decreasing as we can see in Figure 10.8, right. When the points $x^{(k)}$ are close to $-3/2$, the second Wolfe's condition (7.43) is not satisfied as $f'(x^{(k+1)})$ (with its own sign) should be larger than $\delta f'(x^{(k)})$.

Exercise 7.6 After the following initializations

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
grad=@(x) [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
             200*(x(2)-x(1)^2)];
hess=@(x) [-400*x(2)+1200*x(1)^2+2, -400*x(1);
            -400*x(1), 200];
x0=[-1.2,1]; tol=1.e-8; kmax=500;
```

we call Program 7.3 using the following instructions:

```
meth=1; % Newton
[x1,err1,k1]=descent(fun,grad,x0,tol,kmax,meth,hess);
meth=2; H0=eye(length(x0)); % BFGS
[x2,err2,k2]=descent(fun,grad,x0,tol,kmax,meth,H0);
meth=3; %gradient
[x3,err3,k3]=descent(fun,grad,x0,tol,kmax,meth);
meth=41; % FR  conjugate gradient
[x41,err41,k41]=descent(fun,grad,x0,tol,kmax,meth);
meth=42; % PR conjugate gradient
[x42,err42,k42]=descent(fun,grad,x0,tol,kmax,meth);
meth=43; % HS conjugate gradient
[x43,err43,k43]=descent(fun,grad,x0,tol,kmax,meth);
```

All the methods converge to the same global minimizer $(1, 1)$, precisely:

```
Newton:  k1  =  22,  err = 1.8652e-12
BFGS:    k2  =  35,  err = 1.7203e-09
Grad:    k3  = 352,  err = 8.1954e-09
CG-FR:   k41 = 284,  err = 5.6524e-10
```

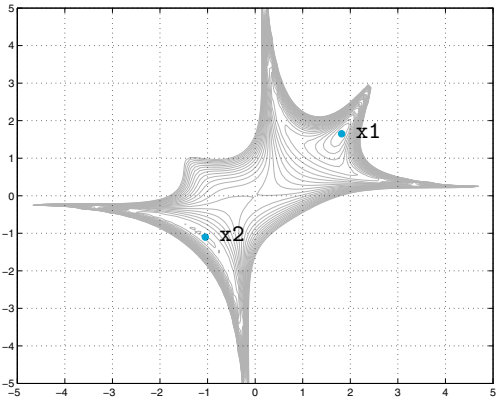

Figure 10.9. Contour lines comprised between the values 0 and 20 of the cost function of Exercise 7.7

```
CG-PR:   k42 = 129,  err = 5.8148e-09
CG-HS:   k43 =  65,  err = 9.8300e-09
```

The number of iterations (`k1, k2, ..., k43`) is in accordance with the theoretical convergence rate of the various methods: quadratic for Newton, superlinear for BFGS, linear for the others. The variable `err` contains the last value of the error estimation used for the stopping test.

Exercise 7.7 By evaluating the function $f(\mathbf{x})$ on the square $[-5,5]^2$ and graphically representing the contour lines corresponding to the values within the interval $[0,20]$, we see that it features a saddle point near $(0,0)$ and two local minimizers, one (`x2`) close to $(-1,-1)$, the other (`x1`) to $(2,2)$ (see Figure 10.9). (One of them will coincide with the global minimizer we are looking for.) Using `tol=1.e-5` as tolerance for the stopping test and 100 as maximum number of iterations, we take `delta0=0.5` as initial radius for the trust region method implemented in Program 7.4. After having defined the function handle of the cost function and its gradient, we set `meth=2` for both Programs 7.4 and 7.3 in such a way that they use quasi-Newton descent directions (and `hess=eye(2)`). Choosing $\mathbf{x}_0 = (2,-1)$, the trust-region method converges in 28 iterations to the point `x1=(1.8171, 1.6510)`, while the BFGS method converges in 27 iterations to the other local minimizer `x2=(-5.3282e-01, -5.8850e-01)`. Correspondingly, $f(\mathtt{x1}) \simeq 3.6661$ and $f(\mathtt{x2}) \simeq 8.2226$. Taking instead $\mathbf{x}^{(0)} = (2,1)$, both methods converge to the global minimizer `x1` in 11 iterations.

Exercise 7.8 Computing the stationary points of $\tilde{f}_k(\mathbf{x}) = \frac{1}{2}\|\widetilde{\mathbf{R}}_k(\mathbf{x})\|^2$ amounts to solve the linear system

$$\nabla \tilde{f}_k(\mathbf{x}) = \mathrm{J}_{\widetilde{\mathbf{R}}_k}(\mathbf{x})^T \widetilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{0}. \tag{10.4}$$

Thanks to the definition (7.64), $\mathrm{J}_{\tilde{\mathbf{R}}_k}(\mathbf{x}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ for all $\mathbf{x} \in \mathbb{R}^n$ and system (10.4) becomes

$$\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0},$$

that is (7.63).

Exercise 7.9 We must show that $\boldsymbol{\delta}\mathbf{x}^{(k)}$ fulfills conditions (7.34). We recall that for every rectangular matrix A having full rank, the square matrix $\mathrm{A}^T\mathrm{A}$ is symmetric and positive definite.

Let us prove $(7.34)_2$. From $\nabla f(\mathbf{x}^{(k)}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)})$, it follows that $\nabla f(\mathbf{x}^{(k)}) = \mathbf{0}$ iff $\mathbf{R}(\mathbf{x}^{(k)}) = \mathbf{0}$ (as $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ has full rank) then $\boldsymbol{\delta}\mathbf{x}^{(k)} = \mathbf{0}$ thanks to $(7.63)_1$.

Suppose now that $\mathbf{R}(\mathbf{x}^{(k)}) \neq \mathbf{0}$. Then

$$\begin{aligned}
&(\boldsymbol{\delta}\mathbf{x}^{(k)})^T \nabla f(\mathbf{x}^{(k)}) = \\
&- \left\{ \left[\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)}) \right]^{-1} \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \right\}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \\
&- \left(\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \right)^T \left[\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)}) \right]^{-1} \left(\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) \right) < 0,
\end{aligned}$$

that is $(7.34)_1$ is fulfilled.

Exercise 7.10 Setting $r_i(\mathbf{x}) = x_1 + x_2 t_i + x_3 t_i^2 + x_4 e^{-x_5 t_i} - y_i$, for $i = 1, \ldots, 8$, the desired coefficients $x_1, \ldots, x_5$, are those for which the associated function (7.61) attains its minimum. We call Program 7.5 using the following instructions:

```
t= [0.055;0.181;0.245;0.342;0.419;0.465;0.593;0.752];
y= [2.80;1.76;1.61;1.21;1.25;1.13;0.52;0.28];
tol=1.e-12; kmax=500;
x0=[2,-2.5,-.2,5,35];
[x,err,iter]= gaussnewton(@mqnlr,@mqnljr,...
    x0,tol,kmax,t,y);
```

where `mqnlr` and `mqnljr` are the functions which define $\mathbf{R}(\mathbf{x})$ and $\mathrm{J}_{\mathbf{R}}(\mathbf{x})$ respectively:

```
function r=mqnlr(x,t,y)
m=length(t); n=length(x);
r=zeros(m,1);
for i=1:m
r(i)=sqrt(2)*(x(1)+t(i)*x(2)+t(i)^2*x(3)+...
    x(4)*exp(-t(i)*x(5))-y(i));
end

function jr=mqnljr(x,t,y)
m=length(t); n=length(x); jr=zeros(m,n);
for i=1:m
jr(i,1)=1; jr(i,2)=t(i);
jr(i,3)=t(i)^2; jr(i,4)=exp(-t(i)*x(5));
jr(i,5)=-t(i)*x(4)*exp(-t(i)*x(5));
end
jr=jr*sqrt(2);
```

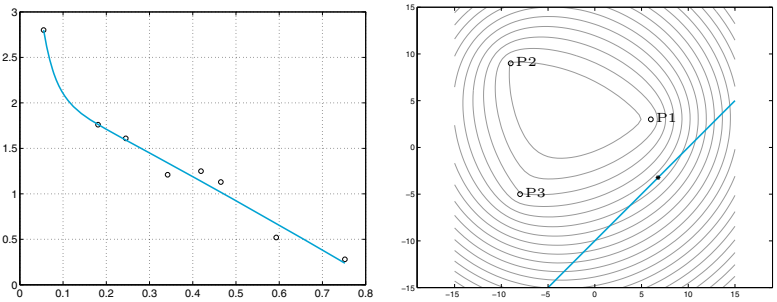

Figure 10.10. Left: the data and the solution of the Exercise 7.10. Right: the solution of the Exercise 7.12. The grey curves represent the contour lines of the cost function. The admissibility domain Ω is the unbounded portion of the plane beneath the blue straight line

After 19 iterations convergence is achieved to the point
`x=[2.2058e+00 -2.4583e+00 -2.1182e-01 5.2106e+00 3.5733e+01]`.
At that point the residual is far from zero, actually $f(x) = 1.8428e-01$. Nevertheless we can classify the given problem as a small residual problem, therefore convergence is linear. For the same problem, Newton's method (7.31) converges in 8 iterations. If the initial point is not close enough to the minimizer, e.g. if `x0 = [1,1,1,1,10]`, the Gauss-Newton method fails to converge, while the damped Gauss-Newton method converges in 21 iterations. In Figure 10.10, left, we plot function $\phi(t)$ whose coefficients $x_1, \ldots, x_5$ are those computed numerically. The empty circles represent the distribution of data (t_i, y_i).

Exercise 7.11 Starting from $\Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{R}(\mathbf{x})\|^2$ a quadratic approximation of Φ around $\mathbf{x}^{(k)}$ reads

$$\tilde{\Phi}_k(\mathbf{s}) = \Phi(\mathbf{x}^{(k)}) + \mathbf{s}^T \nabla\Phi(\mathbf{x}^{(k)}) + \frac{1}{2}\mathbf{s}^T \mathrm{H}_k \mathbf{s} \qquad \forall \mathbf{s} \in \mathbb{R}^n,$$

where H_k is a suitable approximation of the Hessian of Φ. By exploiting (7.62) and taking

$$\mathrm{H}_k = \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)}),$$

it holds

$$\begin{aligned}\tilde{\Phi}_k(\mathbf{s}) &= \frac{1}{2}\|\mathbf{R}(\mathbf{x}^{(k)})\|^2 + \mathbf{s}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathbf{R}(\mathbf{x}^{(k)}) + \frac{1}{2}\mathbf{s}^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s} \\ &= \frac{1}{2}\|\mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s}\|^2 \\ &= \frac{1}{2}\|\tilde{\mathbf{R}}_k(\mathbf{x})\|^2.\end{aligned}$$

In conclusion, $\tilde{\Phi}_k$ can be regarded as a quadratic model of Φ around $\mathbf{x}^{(k)}$, obtained by replacing $\mathbf{R}(\mathbf{x})$ with $\tilde{\mathbf{R}}_k(\mathbf{x})$.

Exercise 7.12 We need to solve the minimization problem (7.2) with cost function $f(x,y) = \sum_{i=1}^{3} v_i \sqrt{(x-x_i)^2+(y-y_i)^2}$ and admissibility domain $\Omega = \{(x,y) \in \mathbb{R}^2 : \ y \leq x - 10\}$. The values v_i represent the number of journeys toward the selling point P_i.
We define first the cost function and the constraint functions, then we call Program `penalty.m`, using the following instructions:

```
x1=[6; 3]; x2=[-9;9]; x3=[-8;-5]; v=[140;134;88];
d=@(x)v(1)*sqrt((x(1)-x1(1)).^2+(x(2)-x1(2)).^2)+...
    v(2)*sqrt((x(1)-x2(1)).^2+(x(2)-x2(2)).^2)+...
    v(3)*sqrt((x(1)-x3(1)).^2+(x(2)-x3(2)).^2);
g=@(x)[x(1)-x(2)-10];
meth=0; x0=[10;-10]; tol=1.e-8; kmax=200;kmaxd=200;
[xmin,err,k]=penalty(d,[],[],[],g,[],x0,tol,...
                kmax,kmaxd,meth);
```

This program makes use of the penalty algorithm coupled with the Nelder and Mead method for unconstrained minimization. We have not used descent method since the cost function features non-differentiable points, moreover the matrices H_k used for the direction $\mathbf{d}^{(k)}$ may be ill-conditioned. The optimal location where to place the warehouse has coordinates `xmin=[6.7734,-3.2266]`. Convergence is achieved after 13 iterations of the penalty method.

Exercise 7.13 Since no inequality constraint is present, problem can be rewritten under the form (7.77) and then we can proceed as done in Example 7.14. Matrix C has rank 2 and its kernel $ker(\mathrm{C}) = \{\mathbf{z} = \alpha[1,1,1]^T \alpha \in \mathbb{R}\}$ has dimension 1. Matrix A is symmetric; as $\sum_{i,j=1}^{3} a_{ij} > 0$, it is positive definite when restricted to the kernel of C. We built matrix $\mathrm{M} = [\mathrm{A}, \ -\mathrm{C}^T; \mathrm{C}, \ 0]$ and the right hand side $\mathbf{f} = [-\mathbf{b}, \mathbf{d}]^T$, then we solve the linear system (7.77) using the instructions:

```
A=[2,-1,1;-1,3,4;1,4,1]; b=[1;-2;-1];
C=[2,-2,0;2,1,-3]; d=[1;1];
M=[A -C'; C, zeros(2)]; f=[-b;d];
xl=M\f;
```

We obtain the solution

```
xl =
   5.7143e-01
   7.1429e-02
   7.1429e-02
   1.0476e+00
   2.3810e-02
```

The first 3 components of `xl` provide the approximation of the minimizer, whereas the Lagrangian multipliers associated to the constraints are given by the last components. The minimum value attained by the cost function is `6.9388e-01`.

Exercise 7.14 We represent the function $v(x,y)$ on the square $[-2.5, 2.5]^2$ and its restriction to the curve $h(x,y) = x^2/4 + y^2 - 1 = 0$ representing the constraint in Figure 10.11. As we can see, several local maximizers exist, the global one lying in a neighborhoud of the point (2,0.5).
We use the following instructions to call Program 7.7

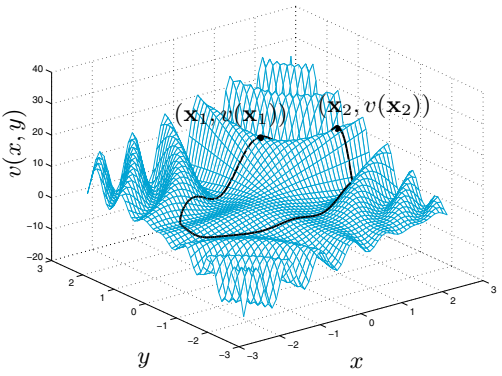

Figure 10.11. The function $v(x, y)$ of Exercise 7.14 and the two maxima computed using the augmented Lagrangian method

```
fun=@(x)-(sin(pi*x(1)*x(2))+1)*(2*x(1)+3*x(2)+4);
grad_fun=@(x)[-pi*x(2)*cos(pi*x(1)*x(2))*...
    (2*x(1)+3*x(2)+4)-(sin(pi*x(1)*x(2))+1)*2;
    -pi*x(1)*cos(pi*x(1)*x(2))*(2*x(1)+3*x(2)+4)-...
    (sin(pi*x(1)*x(2))+1)*3];
h=@(x)x(1)^2/4+x(2)^2-1; grad_h=@(x)[x(1)/2;2*x(2)];
x0=[1;0]; lambda0=1; tol=1.e-8; kmax=100; kmaxd=100;
meth=2;hess=eye(2);
[x,err,k]=auglagrange(fun,grad_fun,h,grad_h,...
     x0,lambda0,tol,kmax,kmaxd,meth,hess)
```

To solve the unconstrained minimization problem for the function $f(x, y) = -v(x, y)$ inside the augmented Lagrangian method, we use BFGS method. Choosing $\mathbf{x}^{(0)} = (1, 0)$, convergence is achieved in 6 iterations to the point $\mathbf{x}_1 = (0.56833, 0.95877)$. The latter is a maximizer but not the global one, as Figure 10.11 shows. Choosing instead $\mathbf{x}^{(0)} = (2, 1)$ we obtain convergence (in 5 iterations) to the point $\mathbf{x}_2 = (1.9242, 0.27265)$; note that $v(\mathbf{x}_1) = 15.94$ while $v(\mathbf{x}_2) = 17.307$, $\mathbf{x}_2$ is therefore the global maximizer.

10.8 Chapter 8

Exercise 8.1 Let us approximate the exact solution $y(t) = \frac{1}{2}[e^t - \sin(t) - \cos(t)]$ of the Cauchy problem (8.85) by the forward Euler method using different values of h: 1/2, 1/4, 1/8, ..., 1/512. The associated error is computed by the following instructions:

```
t0=0; y0=0; T=1; f=@(t,y) sin(t)+y;
y=@(t) 0.5*(exp(t)-sin(t)-cos(t));
Nh=2;
for k=1:10;
[tt,u]=feuler(f,[t0,T],y0,Nh);
e(k)=max(abs(u-y(tt)));Nh=2*Nh;
```

```
end
```

Now we apply formula (1.12) to estimate the order of convergence:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    0.7696    0.9273    0.9806    0.9951    0.9988
```

As expected the order of convergence is one. With the same instructions (substituting the call to Program 8.1 with that to Program 8.2) we obtain an estimate of the convergence order of the backward Euler method:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    1.5199    1.0881    1.0204    1.0050    1.0012
```

Exercise 8.2 The numerical solution of the given Cauchy problem by the forward Euler method can be obtained as follows:

```
t0=0; T=1; N=100; f=@(t,y) -t*exp(-y);
y0=0;[t,u]=feuler(f,[t0,T],y0,N);
```

To compute the number of exact significant digits we can estimate the constants L and M which appear in (8.13). Note that, since $f(t, y(t)) < 0$ in the given interval, $y(t)$ is a monotonically decreasing function, vanishing at $t = 0$. Since f is continuous together with its first derivative, we can approximate L as $L = \max_{0\le t\le 1} |L(t)|$ with $L(t) = \partial f/\partial y = te^{-y}$. Note that $L(0) = 0$ and $L'(t) > 0$ for all $t \in (0, 1]$. Thus, by using the assumption $-1 < y < 0$, we can take $L = e$.

Similarly, in order to compute $M = \max_{0\le t\le 1} |y''(t)|$ with $y'' = -e^{-y} - t^2e^{-2y}$, we can observe that this function has its maximum at $t = 1$, and then $M = e + e^2$. We can draw these conclusions by analyzing the graph of the vector field $\mathbf{v}(t, y) = [v_1, v_2]^T = [1, f(t, y(t))]^T$ associated to the given Cauchy problem. Indeed, the solutions of the differential equation $y'(t) = f(t, y(t))$ are tangential to the vector field $\mathbf{v}$. By the following instructions:

```
[T,Y]=meshgrid(0:0.05:1,-1:0.05:0);
V1=ones(size(T)); V2=-T.*exp(Y); quiver(T,Y,V1,V2)
```

we see that the solution of the Cauchy problem has a nonpositive second derivative whose absolute value grows up with t. This fact leads us to conclude that $M = \max_{0\le t\le 1} |y''(t)|$ is reached at $t = 1$.
The same conclusions can be drawn by noticing that the function $-y$ is positive and increasing, since $y \in [-1, 0]$ and $f(t, y) = y' < 0$. Thus, also the functions e^{-y} and t^2e^{-2y} are positive and increasing, while the function $y'' = -e^{-y} - t^2e^{-2y}$ is negative and decreasing. It follows that $M = \max_{0\le t\le 1} |y''(t)|$ is obtained at $t = 1$.

From (8.13), for $h = 0.01$ we deduce

$$|u_{100} - y(1)| \le \frac{e^L - 1}{L}\frac{M}{200} \simeq 0.26.$$

Therefore, there is no guarantee that more than one significant digit be exact. Indeed, we find `u(end)=-0.6785`, while the exact solution ($y(t) = \log(1-t^2/2)$) at $t = 1$ is $y(1) = -0.6931$.

Exercise 8.3 The iteration function is $\phi(u) = u - ht_{n+1}e^{-u}$ and the fixed-point iteration converges if $|\phi'(u)| < 1$. This property is ensured if $h(t_0 + (n+1)h) < e^u$. If we substitute u with the exact solution, we can provide an *a priori* estimate of the value of h. The most restrictive situation occurs when $u = -1$ (see Solution 8.2). In this case the solution of the inequality $(n+1)h^2 < e^{-1}$ is $h < \sqrt{e^{-1}/(n+1)}$.

Exercise 8.4 We repeat the same set of instructions of Solution 8.1, however now we use the program `cranknic` (Program 8.3) instead of `feuler`. According to the theory, we obtain the following result that shows second-order convergence:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
p =
    2.0379    2.0023    2.0001    2.0000    2.0000
```

Exercise 8.5 Consider the integral formulation of the Cauchy problem (8.5) in the interval $[t_n, t_{n+1}]$:

$$\begin{aligned} y(t_{n+1}) - y(t_n) &= \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau \\ &\simeq \frac{h}{2}\left[f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))\right], \end{aligned}$$

where we have approximated the integral by the trapezoidal formula (4.19). By setting $u_0 = y(t_0)$ and defining u_{n+1} as

$$u_{n+1} = u_n + \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right], \qquad \forall n \geq 0,$$

we obtain precisely the Crank-Nicolson method.

Exercise 8.6 We know that the absolute stability region for the forward Euler method is the circle centered at $(-1, 0)$ with radius equal to 1, that is the set $A = \{z = h\lambda \in \mathbb{C} : \ |1 + h\lambda| < 1\}$. By replacing $\lambda = -1 + i$ we obtain the bound on h: $h^2 - h < 0$, i.e. $h \in (0, 1)$.

Exercise 8.7 Let us rewrite the Heun method in the following (Runge-Kutta like) form:

$$\begin{aligned} &u_{n+1} = u_n + \frac{h}{2}(K_1 + K_2), \\ &K_1 = f(t_n, u_n), \quad K_2 = f(t_{n+1}, u_n + hK_1). \end{aligned} \tag{10.5}$$

We have $h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - h(\widehat{K}_1 + \widehat{K}_2)/2$, with $\widehat{K}_1 = f(t_n, y(t_n))$ and $\widehat{K}_2 = f(t_{n+1}, y(t_n) + h\widehat{K}_1)$. Since f is continuous with respect to both arguments, it holds

$$\lim_{h\to 0} \tau_{n+1} = y'(t_n) - \frac{1}{2}[f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0.$$

Therefore, the Heun method is consistent. We prove now that τ_{n+1} is an infinitesimal of second order with respect to h. Suppose that $y \in C^3([t_0, T[)$. For simplicity of notations, we set $y_n = y(t_n)$ for any $n \geq 0$. We have

$$\begin{aligned}\tau_{n+1} &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))\right] \\ &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}y'(t_n) - \frac{1}{2}f(t_{n+1}, y_n + hy'(t_n)).\end{aligned}$$

Thanks to the error formula (4.20) related to the trapezoidal rule there exists $\xi_n \in]t_n, t_{n+1}[$ such that

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} y'(t)dt = \frac{h}{2}\left[y'(t_n) + y'(t_{n+1})\right] - \frac{h^3}{12}y'''(\xi_n),$$

therefore

$$\begin{aligned}\tau_{n+1} &= \frac{1}{2}\left(y'(t_{n+1}) - f(t_{n+1}, y_n + hy'(t_n)) - \frac{h^2}{6}y'''(\xi_n)\right) \\ &= \frac{1}{2}\left(f(t_{n+1}, y_{n+1}) - f(t_{n+1}, y_n + hy'(t_n)) - \frac{h^2}{6}y'''(\xi_n)\right).\end{aligned}$$

Moreover, as the function f is Lipschitz continuous with respect to the second variable (see Proposition 8.1), it holds

$$|\tau_{n+1}| \leq \frac{L}{2}|y_{n+1} - y_n - hy'(t_n)| + \frac{h^2}{12}|y'''(\xi_n)|.$$

Finally, by applying the Taylor formula

$$y_{n+1} = y_n + hy'(t_n) + \frac{h^2}{2}y''(\eta_n), \qquad \eta_n \in]t_n, t_{n+1}[,$$

we obtain

$$|\tau_{n+1}| \leq \frac{L}{4}h^2|y''(\eta_n)| + \frac{h^2}{12}|y'''(\xi_n)| \leq Ch^2.$$

The Heun method is implemented in Program 10.2. Using this program, we can verify the order of convergence as in Solution 8.1. Precisely, by the following instructions, we find that the Heun method is second-order accurate with respect to h

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
ans =
    1.7642    1.9398    1.9851    1.9963    1.9991
```

Program 10.2. rk2: Heun (or RK2) method

```
function [tt,u]=rk2(odefun,tspan,y0,Nh,varargin)
tt=linspace(tspan(1),tspan(2),Nh+1);
h=(tspan(2)-tspan(1))/Nh;  hh=h*0.5;
u=y0;
for t=tt(1:end-1)
  y = u(end,:);
  k1=odefun(t,y,varargin{:});
```

```
  t1 = t + h; y = y + h*k1;
  k2=odefun(t1,y,varargin{:});
  u = [u; u(end,:) + hh*(k1+k2)];
end
tt=tt';
```

Exercise 8.8 Applying the method (10.5) to the model problem (8.28) we obtain $K_1 = \lambda u_n$ and $K_2 = \lambda u_n(1+h\lambda)$. Therefore $u_{n+1} = u_n[1+h\lambda+(h\lambda)^2/2] = u_n p_2(h\lambda)$. To ensure absolute stability we must require that $|p_2(h\lambda)| < 1$, which is equivalent to $0 < p_2(h\lambda) < 1$, since $p_2(h\lambda)$ is positive. Solving the latter inequality, we obtain $-2 < h\lambda < 0$, that is, $h < 2/|\lambda|$, since λ is a real negative number.

Exercise 8.9 We prove the property (8.34), that we call for simplicity $\mathcal{P}_n$, by induction on n. To this aim, it is sufficient to prove that if $\mathcal{P}_1$ holds and if $\mathcal{P}_{n-1}$ implies $\mathcal{P}_n$ for any $n \geq 2$, then $\mathcal{P}_n$ holds for any $n \geq 2$.
It is easily verified that $u_1 = u_0 + h(\lambda_0 u_0 + r_0)$. In order to prove that $\mathcal{P}_{n-1} \Rightarrow \mathcal{P}_n$, it is sufficient to note that $u_n = u_{n-1}(1 + h\lambda_{n-1}) + hr_{n-1}$.

Exercise 8.10 Since $|1 + h\lambda| < 1$, from (8.38) it follows

$$|z_n - u_n| \leq |\rho| \left(\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| \right).$$

If $\lambda \leq -1$, we have $1/\lambda < 0$ and $1 + 1/\lambda \geq 0$, then

$$\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| = 1 + \frac{1}{\lambda} - \frac{1}{\lambda} = 1 = \varphi(\lambda).$$

On the other hand, if $-1 < \lambda < 0$, we have $1/\lambda < 1 + 1/\lambda < 0$, then

$$\left| 1 + \frac{1}{\lambda} \right| + \left| \frac{1}{\lambda} \right| = -1 - \frac{2}{\lambda} = \left| 1 + \frac{2}{\lambda} \right| = \varphi(\lambda).$$

Exercise 8.11 From (8.36) we have

$$|z_n - u_n| \leq \overline{\rho}[a(h)]^n + h\overline{\rho} \sum_{k=0}^{n-1} [a(h)]^{n-k-1}.$$

The result follows using (8.37).

Exercise 8.12 We have

$$h\tau_{n+1}(h) = y(t_{n+1}) - y(t_n) - \frac{h}{6}(\widehat{K}_1 + 4\widehat{K}_2 + \widehat{K}_3),$$
$$\widehat{K}_1 = f(t_n, y(t_n)), \quad \widehat{K}_2 = f(t_n + \tfrac{h}{2}, y(t_n) + \tfrac{h}{2}\widehat{K}_1),$$
$$\widehat{K}_3 = f(t_{n+1}, y(t_n) + h(2\widehat{K}_2 - \widehat{K}_1)).$$

Since f is continuous with respect to both arguments, we obtain

$$\lim_{h\to 0}\tau_{n+1} = y'(t_n) - \frac{1}{6}[f(t_n, y(t_n)) + 4f(t_n, y(t_n)) + f(t_n, y(t_n))] = 0,$$

which proves that the method is consistent.

This method is an explicit Runge-Kutta method of order 3 and is implemented in Program 10.3. As in Solution 8.7, we can derive an estimate of its order of convergence by the following instructions:

```
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)
ans =
    2.7306    2.9330    2.9833    2.9958    2.9990
```

Program 10.3. rk3: explicit Runge-Kutta method of order 3

```
function [tt,u]=rk3(odefun,tspan,y0,Nh,varargin);
tt=linspace(tspan(1),tspan(2),Nh+1);
h=(tspan(2)-tspan(1))/Nh; hh=h*0.5; h2=2*h;
u=y0; h6=h/6;
for t=tt(1:end-1)
  y = u(end,:);
  k1=odefun(t,y,varargin{:});
  t1 = t + hh; y1 = y + hh* k1;
  k2=odefun(t1,y1,varargin{:});
  t1 = t + h; y1 = y + h*(2*k2-k1);
  k3=odefun(t1,y1,varargin{:});
  u = [u; u(end,:) + h6*(k1+4*k2+k3)];
end
tt=tt';
```

Exercise 8.13 By following the same arguments used in Solution 8.8, we obtain the relation

$$u_{n+1} = u_n[1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3] = u_n p_3(h\lambda).$$

By inspection of the graph of p_3, obtained with the instruction

```
c=[1/6 1/2 1 1]; z=[-3:0.01:1];
p=polyval(c,z); plot(z,abs(p))
```

we deduce that $|p_3(h\lambda)| < 1$, provided that $-2.5 < h\lambda < 0$.

Exercise 8.14 The method (8.87) applied to the model problem (8.28) with $\lambda \in \mathbb{R}^-$ gives the equation $u_{n+1} = u_n(1+h\lambda+(h\lambda)^2)$. By solving the inequality $|1 + h\lambda + (h\lambda)^2| < 1$ we find $-1 < h\lambda < 0$.

Exercise 8.15 To solve Problem 8.1 with the given values, we repeat the following instructions with `N=10` and `N=20`:

```
f=@(t,y) -1.68e-9*y^4+2.6880;
[tc,uc]=cranknic(f,[0,200],180,N);
[tp,up]=rk2(f,[0,200],180,N);
```

The graphs of the computed solutions are shown in Figure 10.12.

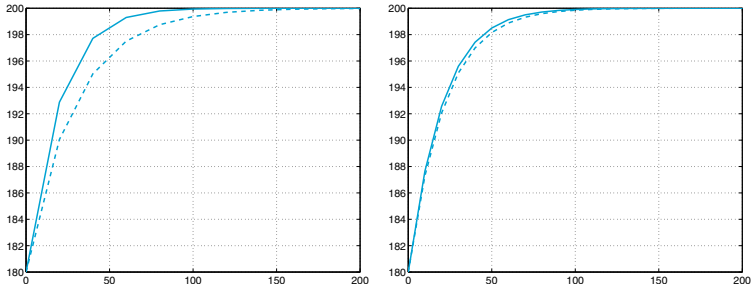

Figure 10.12. Computed solutions with `N = 10` (*left*) and `N = 20` (*right*) for the Cauchy problem of Solution 8.15: the solutions computed by the Crank-Nicolson method (*solid line*), and by the Heun method (*dashed line*)

Exercise 8.16 Heun method applied to the model problem (8.28), gives

$$u_{n+1} = u_n \left(1 + h\lambda + \frac{1}{2}h^2\lambda^2\right).$$

In the complex plane the boundary of the region of absolute stability is the set of points $h\lambda = x+iy$ such that $|1+h\lambda+h^2\lambda^2/2|^2 = 1$. This equation is satisfied by the pairs (x, y) such that $f(x, y) = x^4+y^4+2x^2y^2+4x^3+4xy^2+8x^2+8x = 0$. We can represent this curve as the 0-contour line of the function $z = f(x, y)$. This can be done by means of the following instructions:

```
f=@(x,y)[x.^4+y.^4+2*(x.^2).*(y.^2)+...
         4*x.*y.^2+4*x.^3+8*x.^2+8*x];
[x,y]=meshgrid([-2.1:0.1:0.1],[-2:0.1:2]);
contour(x,y,f(x,y),[0 0]); grid on
```

The command `meshgrid` draws in the rectangle $[-2.1, 0.1] \times [-2, 2]$ a grid with 23 equispaced nodes in the x-direction, and 41 equispaced nodes in the y-direction. With the command `contour` we plot the contour line of $f(x, y)$ corresponding to the value $z = 0$ (made precise in the input vector `[0 0]` of `contour`). In Figure 10.13 the solid line delimitates the region of absolute stability of the Heun method. This region is larger than the absolute stability region of the forward Euler method (which corresponds to the interior of the dashed circle). Both curves are tangent to the imaginary axis at the origin $(0, 0)$.

`contour`

Exercise 8.17 We use the following instructions:

```
t0=0; y0=0; f=@(t,y)cos(2*y);
y=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
T=1; N=2; for k=1:10;
[tt,u]=rk2(f,[t0,T],y0,N);
e(k)=max(abs(u-y(tt))); N=2*N; end
p=log(abs(e(1:end-1)./e(2:end)))/log(2); p(1:2:end)

    2.4733    2.1223    2.0298    2.0074    2.0018
```

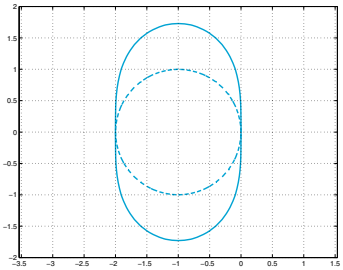

Figure 10.13. Boundaries of the regions of absolute stability for the Heun method (*solid line*) and the forward Euler method (*dashed line*). The corresponding regions lie at the interior of the boundaries

As expected, we find that the order of convergence of the method is 2. However, the computational cost is comparable with that of the forward Euler method, which is first-order accurate only.

Exercise 8.18 The second-order differential equation of this exercise is equivalent to the following first-order system:

$$x'(t) = z(t), \quad z'(t) = -5z(t) - 6x(t),$$

with $x(0) = 1$, $z(0) = 0$. We use the Heun method as follows:

```
t0=0; y0=[1 0]; T=5;
[t,u]=rk2(@fspring,[t0,T],y0,N);
```

where `N` is the number of nodes and `fspring.m` is the following function:

```
function fn=fspring(t,y)
b=5;
k=6;
[n,m]=size(y);
fn=zeros(n,m);
fn(1)=y(2);
fn(2)=-b*y(2)-k*y(1);
```

In Figure 10.14 we show the graphs of the two components of the solution, computed with `N=20` and `N=40` and compare them with the graph of the exact solution $x(t) = 3e^{-2t} - 2e^{-3t}$ and that of its first derivative.

Exercise 8.19 The second-order system of differential equations is reduced to the following first-order system:

$$\begin{cases} x'(t) = z(t), \\ y'(t) = v(t), \\ z'(t) = 2\omega \sin(\Psi) v(t) - k^2 x(t), \\ v'(t) = -2\omega \sin(\Psi) z(t) - k^2 y(t). \end{cases} \tag{10.6}$$

If we suppose that the pendulum at the initial time $t_0 = 0$ is at rest in the position $(1, 0)$, the system (10.6) must be given the following initial conditions:

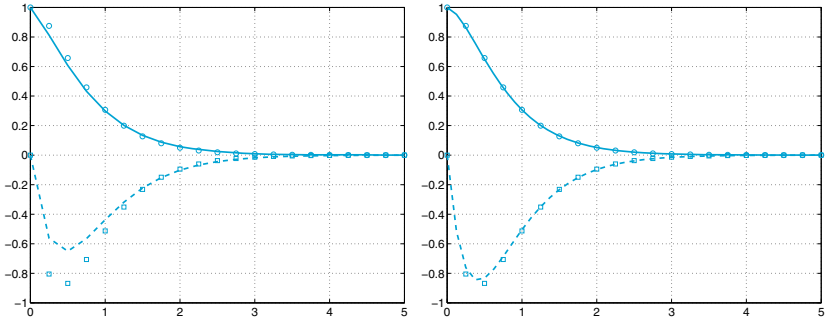

Figure 10.14. Approximations of $x(t)$ (*solid line*) and $x'(t)$ (*dashed line*) computed with `N=20` (*at left*) and `N=40` (*at right*). Small circles and squares refer to the exact functions $x(t)$ and $x'(t)$, respectively

$$x(0) = 1,\ y(0) = 0,\ z(0) = 0,\ v(0) = 0.$$

Setting $\Psi = \pi/4$, which is the average latitude of the Northern Italy, we use the forward Euler method as follows:

```
[t,u]=feuler(@ffoucault,[0,300],[1 0 0 0],N);
```

where `N` is the number of steps and `ffoucault.m` is the following function:

```
function fn=ffoucault(t,y)
l=20; k2=9.8/l; psi=pi/4; omega=7.29*1.e-05;
[n,m]=size(y); fn=zeros(n,m);
fn(1)=y(3);  fn(2)=y(4);
fn(3)=2*omega*sin(psi)*y(4)-k2*y(1);
fn(4)=-2*omega*sin(psi)*y(3)-k2*y(2);
```

By some numerical experiments we conclude that the forward Euler method cannot produce acceptable solutions for this problem even for very small h. For instance, on the left of Figure 10.15 we show the graph, in the phase plane (x, y), of the motion of the pendulum computed with `N=30000`, that is, $h = 1/100$. As expected, the rotation plane changes with time, but also the amplitude of the oscillations increases. Similar results can be obtained for smaller h and using the Heun method. In fact, the model problem corresponding to the problem at hand has a coefficient λ that is purely imaginary. The corresponding solution (a sine function) is bounded for any t, however it doesn't tend to zero.

Unfortunately, both the forward Euler and Heun methods feature a region of absolute stability that doesn't include any point of the imaginary axis (with the exception of the origin). Thus, to ensure the absolute stability one should choose the prohibited value $h = 0$.

To get an acceptable solution we should use a method whose region of absolute stability includes a portion of the imaginary axis. This is the case, for instance, for the adaptive Runge-Kutta method of order 3, implemented in the MATLAB function `ode23`. We can invoke it by the following command:

```
[t,u]=ode23(@ffoucault,[0,300],[1 0 0 0]);
```

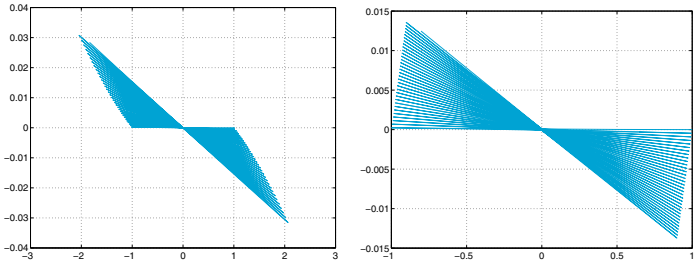

Figure 10.15. Trajectories on the phase plane for the Foucault pendulum of Solution 8.19 computed by the forward Euler method (*left*) and the third-order adaptive Runge-Kutta method (*right*)

In Figure 10.15 (*right*) we show the solution obtained using only 1022 integration steps. Note that the numerical solution is in good agreement with the exact one.

Exercise 8.20 We fix the right hand side of the problem in the following *function*

```
function fn=baseball(t,y)
phi = pi/180;  omega = 1800*1.047198e-01;
B = 4.1*1.e-4; g = 9.8;
[n,m]=size(y); fn=zeros(n,m);
vmodule = sqrt(y(4)^2+y(5)^2+y(6)^2);
Fv = 0.0039+0.0058/(1+exp((vmodule-35)/5));
fn(1)=y(4);
fn(2)=y(5);
fn(3)=y(6);
fn(4)=-Fv*vmodule*y(4)+...
       B*omega*(y(6)*sin(phi)-y(5)*cos(phi));
fn(5)=-Fv*vmodule*y(5)+B*omega*y(4)*cos(phi);
fn(6)=-g-Fv*vmodule*y(6)-B*omega*y(4)*sin(phi);
```

At this point we only need to recall `ode23` as follows:

```
[t,u]=ode23(@baseball,[0 0.4],...
      [0 0 0 38*cos(pi/180) 0 38*sin(pi/180)]);
```

Using command `find` we approximately compute the time at which the altitude becomes negative, which corresponds to the exact time of impact with the ground:

```
n=max(find(u(:,3)>=0)); t(n)
ans =
    0.1066
```

In Figure 10.16 we report the trajectories of the baseball with an inclination of 1 and 3 degrees represented on the plane x_1x_3 and on the $x_1x_2x_3$ space, respectively.

Exercise 8.21 Let us define the function

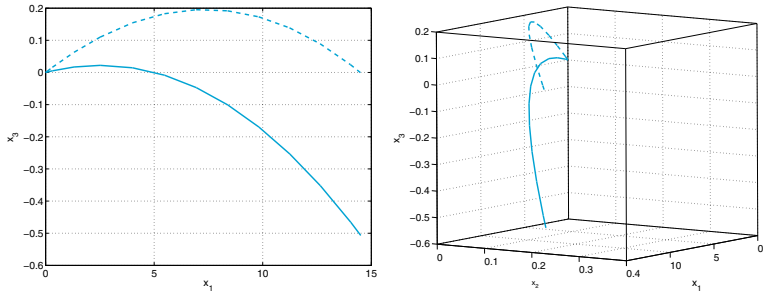

Figure 10.16. The trajectories followed by a baseball launched with an initial angle of 1 degree (*solid line*) and 3 degrees (*dashed line*), respectively

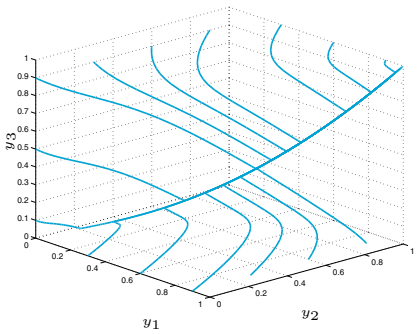

Figure 10.17. Trajectories of the model (8.88) corresponding to several intial data and with $\varepsilon = 10^{-2}$

```
function f=fchem3(t,y)
e=1.e-2;
[n,m]=size(y);f=zeros(n,m);
f(1)=1/e*(-5*y(1)-y(1)*y(2)+5*y(2)^2+...
      y(3))+y(2)*y(3)-y(1);
f(2)=1/e*(10*y(1)-y(1)*y(2)-10*y(2)^2+y(3))...
      -y(2)*y(3)+y(1);
f(3)=1/e*(y(1)*y(2)-y(3))-y(2)*y(3)+y(1);
```

and execute the following instructions

```
y0=[1,0.5,0]; tspan=[0,10];
[t1,y1]=ode23(@fchem3,tspan,y0);
[t2,y2]=ode23s(@fchem3,tspan,y0);
fprintf('Passi ode23=%d, passi ode23s=%d\n',...
length(t1),length(t2))
```

`ode23` requires 8999 steps while `ode23s` only 43. Consequently we can state that the given problem is stiff. The computed numerical solutions are shown in Figure 10.17.

10.9 Chapter 9

Exercise 9.1 We can verify directly that $\mathbf{x}^T \mathrm{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$. Indeed,

$$[x_1\ x_2\ \ldots\ x_{N-1}\ x_N] \begin{bmatrix} 2 & -1 & 0 & \ldots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & -1 & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \ldots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$
$$= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + \ldots - 2x_{N-1}x_N + 2x_N^2.$$

The last expression is equivalent to $(x_1-x_2)^2+\ldots+(x_{N-1}-x_N)^2+x_1^2+x_N^2$, which is positive, provided that at least one x_i is non-null.

Exercise 9.2 We verify that $\mathrm{A}\mathbf{q}_j = \lambda_j \mathbf{q}_j$. Computing the matrix-vector product $\mathbf{w} = \mathrm{A}\mathbf{q}_j$ and requiring that $\mathbf{w}$ is equal to the vector $\lambda_j \mathbf{q}_j$, we find:

$$\begin{cases} 2\sin(j\theta) - \sin(2j\theta) = 2(1-\cos(j\theta))\sin(j\theta), \\ -\sin(j(k-1)\theta) + 2\sin(jk\theta) - \sin(j(k+1)\theta) = 2(1-\cos(j\theta))\sin(kj\theta), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k = 2, \ldots, N-1 \\ 2\sin(Nj\theta) - \sin((N-1)j\theta) = 2(1-\cos(j\theta))\sin(Nj\theta). \end{cases}$$

The first equation is an identity since $\sin(2j\theta) = 2\sin(j\theta)\cos(j\theta)$. The other equations can be simplified in view of the sum-to-product formula

$$\sin((k-1)j\theta) + \sin((k+1)j\theta) = 2\sin(kj\theta)\cos(j\theta)$$

and noticing that $\sin((N+1)j\theta) = 0$ since $\theta = \pi/(N+1)$. Since A is symmetric and positive definite, its condition number is $K(\mathrm{A}) = \lambda_{max}/\lambda_{min}$, that is, $K(\mathrm{A}) = \lambda_N/\lambda_1 = (1-\cos(N\pi/(N+1)))/(1-\cos(\pi/(N+1)))$. By the identity $\cos(N\pi/(N+1)) = -\cos(\pi/(N+1))$ and by using the Taylor expansion of order 2 of the cosine function, we obtain $K(\mathrm{A}) \simeq (N+1)^2$, that is, $K(\mathrm{A}) \simeq h^{-2}$.

Exercise 9.3 We note that

$$u(\bar{x}+h) = u(\bar{x}) + hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) + \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_+),$$
$$u(\bar{x}-h) = u(\bar{x}) - hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u^{(4)}(\xi_-),$$

where $\xi_+ \in (x, x+h)$ and $\xi_- \in (x-h, x)$. Summing the two expression we obtain

$$u(\bar{x}+h) + u(\bar{x}-h) = 2u(\bar{x}) + h^2u''(\bar{x}) + \frac{h^4}{24}(u^{(4)}(\xi_+) + u^{(4)}(\xi_-)),$$

which is the desired property.

Exercise 9.4 The matrix is tridiagonal with entries $a_{i,i-1} = -\mu/h^2 - \eta/(2h)$, $a_{ii} = 2\mu/h^2 + \sigma$, $a_{i,i+1} = -\mu/h^2 + \eta/(2h)$. The right-hand side, accounting for the boundary conditions, becomes $\mathbf{f} = (f(x_1) + \alpha(\mu/h^2 + \eta/(2h)), f(x_2), \ldots, f(x_{N-1}), f(x_N) + \beta(\mu/h^2 - \eta/(2h)))^T$.

Exercise 9.5 With the following instructions we compute the corresponding solutions to the three given values of h:

```
f=@(x) 1+sin(4*pi*x);
[x,uh11]=bvp(0,1,9,1,0,0.1,f,0,0);
[x,uh21]=bvp(0,1,19,1,0,0.1,f,0,0);
[x,uh41]=bvp(0,1,39,1,0,0.1,f,0,0);
```

We recall that $h = (b-a)/(N+1)$. Since we don't know the exact solution, to estimate the convergence order we compute an approximate solution on a very fine grid (for instance $h = 1/1000$), then we use this latter as a surrogate for the exact solution. We find:

```
[x,uhex]=bvp(0,1,999,1,0,0.1,f,0,0);
max(abs(uh11-uhex(1:100:end)))

ans =
   8.6782e-04
max(abs(uh21-uhex(1:50:end)))
ans =
   2.0422e-04
max(abs(uh41-uhex(1:25:end)))
ans =
   5.2789e-05
```

Halving h, the error is divided by 4, proving that the convergence order with respect to h is 2.

Exercise 9.6 We should modify the Program 9.1 in order to impose Neumann boundary conditions. In the Program 10.4 we show one possible implementation.

Program 10.4. neumann: numerical solution of a Neumann boundary-value problem

```
function [xh,uh]=neumann(a,b,N,mu,eta,sigma,bvpfun,...
                         ua,ub,varargin)
h = (b-a)/(N+1); xh = (linspace(a,b,N+2))';
hm = mu/h^2;  hd = eta/(2*h); e =ones(N+2,1);
A = spdiags([-hm*e-hd (2*hm+sigma)*e -hm*e+hd],...
    -1:1, N+2, N+2);
A(1,1)=3/(2*h); A(1,2)=-2/h; A(1,3)=1/(2*h); f(1)=ua;
A(N+2,N+2)=3/(2*h); A(N+2,N+1)=-2/h; A(N+2,N)=1/(2*h);
f =bvpfun(xh,varargin{:});  f(1)=ua; f(N+2)=ub;
uh = A\f;
```

Exercise 9.7 The trapezoidal integration formula, used on the two subintervals I_{j-1} and I_j, produces the following approximation

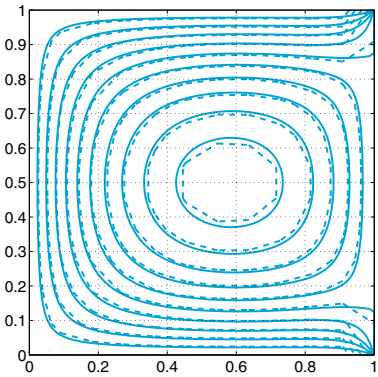

Figure 10.18. The contour lines of the computed temperature for $\Delta_x = \Delta_y = 1/10$ (*dashed lines*) and for $\Delta_x = \Delta_y = 1/80$ (*solid lines*)

$$\int_{I_{j-1}\cup I_j}^{f} (x)\varphi_j(x)\, dx \simeq \frac{h}{2} f(x_j) + \frac{h}{2} f(x_j) = hf(x_j),$$

since $\varphi_j(x_i) = \delta_{ij}$ for any i, j. When $j = 1$ or $j = N$ we can proceed similarly, taking into account the Dirichlet boundary conditions. Thus, we obtain the same right-hand side of the finite difference system (9.14) up to the factor h.

Exercise 9.8 We have $\nabla\phi = (\partial\phi/\partial x, \partial\phi/\partial y)^T$ and therefore $\mathrm{div}\nabla\phi = \partial^2\phi/\partial x^2 + \partial^2\phi/\partial y^2$, that is, the Laplacian of ϕ.

Exercise 9.9 To compute the temperature at the center of the plate, we solve the corresponding Poisson problem for various values of $\Delta_x = \Delta_y$, using the following instructions:

```
k=0; fun=@(x,y) 25+0*x+0*y;
bound=@(x,y) (x==1);
for N = [10,20,40,80,160]
[xh,yh,uh]=poissonfd(0,1,0,1,N,N,fun,bound);
k=k+1; uc(k) = uh(N/2+1,N/2+1);
end
```

The components of the vector `uc` are the values of the computed temperature at the center of the plate as the steplength h of the grid decreases. We have

```
uc
    2.0168    2.0616    2.0789    2.0859    2.0890
```

We can therefore conclude that at the center of the plate the temperature is about 2.08°C. In Figure 10.18 we show the contour lines of the temperature for two different values of h.

Exercise 9.10 For sake of simplicity we set $u_t = \partial u/\partial t$ and $u_x = \partial u/\partial x$. We multiply by u_t the equation (9.72) with $f \equiv 0$, integrate in space on (a, b) and use integration by parts on the second term:

$$\int_a^b u_{tt}(x,t)u_t(x,t)\mathrm{d}x + c\int_a^b u_x(x,t)u_{tx}(x,t)\mathrm{d}x - c[u_x(x,t)u_t(x,t)]_a^b = 0. \tag{10.7}$$

Now we integrate in time equation (10.7), from 0 up to t. By noticing that $u_{tt}u_t = \frac{1}{2}(u_t^2)_t$ and that $u_x u_{xt} = \frac{1}{2}(u_x^2)_t$, by applying the fundamental theorem of integral calculus and recalling the initial conditions (9.74) (that is $u_t(x,0) = v_0(x)$ and $u_x(x,0) = u_{0x}(x)$), we obtain

$$\int_a^b u_t^2(x,t)\mathrm{d}x + c\int_a^b u_x^2(x,t)\mathrm{d}x = \int_a^b v_0^2(x)\mathrm{d}x$$

$$+c\int_a^b u_{0x}^2(x)\mathrm{d}x + 2c\int_0^t (u_x(b,s)u_t(b,s) - u_x(a,s)u_t(a,s))\,\mathrm{d}s.$$

On the other hand, by integrating by parts and applying the homogeneous Dirichlet boundary conditions for $t > 0$ and on the initial data we obtain

$$\int_0^t (u_x(b,s)u_t(b,s) - u_x(a,s)u_t(a,s))\mathrm{d}s = 0.$$

Then (9.83) follows.

Exercise 9.11 In view of definition (9.64) it is sufficient to verify that

$$\sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 \le \sum_{j=-\infty}^{\infty} |u_j^n|^2. \tag{10.8}$$

In formula (9.62), let us move all terms to the left-hand side and then multiply by u_j^{n+1}. Owing to the identity $2(a-b)a = a^2 - b^2 + (a-b)^2$ we have

$$|u_j^{n+1}|^2 - |u_j^n|^2 + |u_j^{n+1} - u_j^n|^2 + \lambda a(u_{j+1}^{n+1} - u_{j-1}^{n+1})u_j^{n+1} = 0,$$

then, summing up on j and noticing that $\sum_{j=-\infty}^{\infty}(u_{j+1}^{n+1} - u_{j-1}^{n+1})u_j^{n+1} = 0$, we obtain

$$\sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 \le \sum_{j=-\infty}^{\infty} |u_j^{n+1}|^2 + \sum_{j=-\infty}^{\infty} |u_j^{n+1} - u_j^n|^2 \le \sum_{j=-\infty}^{\infty} |u_j^n|^2.$$

Exercise 9.12 The *upwind* scheme (9.59) can be rewritten in the simplified form

$$u_j^{n+1} = \begin{cases} (1-\lambda a)u_j^n + \lambda a u_{j-1}^n & \text{if } a > 0 \\ (1+\lambda a)u_j^n - \lambda a u_{j+1}^n & \text{if } a < 0. \end{cases}$$

Let us first consider the case $a > 0$. If the CFL condition is satisfied, then both coefficients $(1-\lambda a)$ and λa are positive and less than 1.
This fact implies that

$$\min\{u_{j-1}^n, u_j^n\} \le u_j^{n+1} \le \max\{u_{j-1}^n, u_j^n\}$$

and, by recursion on n, it holds

$$\inf_{l\in\mathbb{Z}}\{u_l^0\} \le u_j^{n+1} \le \sup_{l\in\mathbb{Z}}\{u_l^0\} \quad \forall n \ge 0,$$

from which the estimate (9.85) follows.
When $a < 0$, using again the CFL condition, both coefficients $(1 + \lambda a)$ and $-\lambda a$ are positive and less than 1. By proceeding as we did before, the estimate (9.85) follows also in this case.

Exercise 9.13 To numerically solve problem (9.47) we call the Program 10.5. Note that the exact solution is the travelling wave with velocity $a = 1$, that is $u(x,t) = 2\cos(4\pi(x-t)) + \sin(20\pi(x-t))$. Since the CFL number is fixed to 0.5, the discretization parameters Δx and Δt are related through the equation $\Delta t = CFL \cdot \Delta x$, thus we can arbitrarily choose only one of them.
In order to verify the accuracy of the scheme with respect to Δt we can use the following instructions:

```
xspan=[0,0.5];
tspan=[0,1];
a=1; cfl=0.5;
u0=@(x) 2*cos(4*pi*x)+sin(20*pi*x);
uex=@(x,t) 2*cos(4*pi*(x-t))+sin(20*pi*(x-t));
ul=@(t) 2*cos(4*pi*t)-sin(20*pi*t);
DT=[1.e-2,5.e-3,2.e-3,1.e-3,5.e-4,2.e-4,1.e-4];
e_lw=[]; e_up=[];
for deltat=DT
deltax=deltat*a/cfl;
[xx,tt,u_lw]=hyper(xspan,tspan,u0,ul,2,...
      cfl,deltax,deltat);
[xx,tt,u_up]=hyper(xspan,tspan,u0,ul,3,...
      cfl,deltax,deltat);
U=uex(xx,tt(end));
[Nx,Nt]=size(u_lw);
e_lw=[e_lw sqrt(deltax)*norm(u_lw(Nx,:)-U,2)];
e_up=[e_up sqrt(deltax)*norm(u_up(Nx,:)-U,2)];
end
p_lw=log(abs(e_lw(1:end-1)./e_lw(2:end)))./...
     log(DT(1:end-1)./DT(2:end))
p_up=log(abs(e_up(1:end-1)./e_up(2:end)))./...
     log(DT(1:end-1)./DT(2:end))

p_lw =
  0.1939   1.8626   2.0014   2.0040   2.0112   2.0239
p_up =
  0.2272   0.3604   0.5953   0.7659   0.8853   0.9475
```

By implementing a similar loop for the parameter Δx, we can verify the accuracy of the scheme with respect to the space discretization. Precisely, for Δx ranging from 10^{-4} to 10^{-2} we obtain

```
p_lw =
  1.8113   2.0235   2.0112   2.0045   2.0017   2.0007
p_up =
  0.3291   0.5617   0.7659   0.8742   0.9407   0.9734
```

Program 10.5. hyper: Lax-Friedrichs, Lax-Wendroff and upwind schemes

```
function [xh,th,uh]=hyper(xspan,tspan,u0,ul,...
                          scheme,cfl,deltax,deltat)
% HYPER solves hyperbolic scalar equations
% [XH,TH,UH]=HYPER(XSPAN,TSPAN,U0,UL,SCHEME,CFL,...
%                  DELTAX,DELTAT)
% solves the hyperbolic scalar equation
%       DU/DT+ A * DU/DX=0
% in (XSPAN(1),XSPAN(2))x(TSPAN(1),TSPAN(2))
% with A>0, initial condition  U(X,0)=U0(X) and
% boundary condition U(T)=UL(T) given at XSPAN(1)
% with several finite difference schemes.
% scheme = 1 Lax - Friedrichs
%          2 Lax - Wendroff
%          3 Upwind
% The propagation velocity 'a' is not required as
% input of the function, since it can be derived
% from CFL = A * DELTAT / DELTAX
% Output: XH is the vector of space nodes
% TH  is the vector of time nodes
% UH is a matrix containing the computed solution
% UH(n,:) contains the solution at time TT(n)
% U0 and UL can be either inline, anonymous
% functions or functions defined by M-file.
Nt=(tspan(2)-tspan(1))/deltat+1;
th=linspace(tspan(1),tspan(2),Nt);
Nx=(xspan(2)-xspan(1))/deltax+1;
xh=linspace(xspan(1),xspan(2),Nx);
u=zeros(Nt,Nx); cfl2=cfl*0.5; cfl21=1-cfl^2;
cflp1=cfl+1; cflm1=cfl-1; uh(1,:)=u0(xh);
for n=1:Nt-1
 uh(n+1,1)=ul(th(n+1));
 if scheme == 1
% Lax Friedrichs
    for j=2:Nx-1
      uh(n+1,j)=0.5*(-cflm1*uh(n,j+1)+cflp1*uh(n,j-1));
    end
    j=Nx;
    uh(n+1,j)=0.5*(-cflm1*(2*uh(n,j)-uh(n,j-1))+...
        cflp1*uh(n,j-1));
 elseif scheme == 2
% Lax Wendroff
    for j=2:Nx-1
     uh(n+1,j)=cfl21*uh(n,j)+...
         cfl2*(cflm1*uh(n,j+1)+cflp1*uh(n,j-1));
    end
    j=Nx;
    uh(n+1,j)=cfl21*uh(n,j)+...
    cfl2*(cflm1*(2*uh(n,j)-uh(n,j-1))+cflp1*uh(n,j-1));
 elseif scheme ==3
% Upwind
   for j=2:Nx
        uh(n+1,j)=-cflm1*uh(n,j)+cfl*uh(n,j-1);
    end
 end
end
```

Exercise 9.14 The exact solution is the sum of two simple harmonics, the former with low frequency and the latter with high frequency. If we choose $\Delta t = 5 \cdot 10^{-2}$, since $a = 1$ and CFL=0.8, we have $\Delta x = 6.25e - 3$ and the phase angles associated to the harmonics are $\phi_{k_1} = 4\pi \cdot 6.25e - 3 \simeq 0.078$ and $\phi_{k_2} = 20\pi \cdot 6.25e - 3 \simeq 0.393$, respectively. By inspecting Figure 9.18 we note that the upwind scheme is more dissipative than Lax-Wendroff's. This fact is confirmed by the behavior of dissipation coefficients (see the right graph at the bottom of Figure 9.14). Indeed, when we take into account instances of ϕ_k corresponding to the given harmonics, the curve relative to the Lax-Wendroff scheme is nearer to the constant 1 than the curve associated to the upwind scheme.

For what concerns the dispersion coefficient, we see from Figure 9.18 that the Lax-Wendroff scheme features a phase delay, while the *upwind* scheme presents a light phase advance. The right graph at the bottom of Figure 9.15 confirms this conclusion. Moreover we can observe that the phase delay of the Lax-Wendroff scheme is larger than the phase advance of the upwind scheme.