8

Ordinary differential equations

A differential equation is an equation involving one or more derivatives of an unknown function. If all derivatives are taken with respect to a single independent variable we call it an *ordinary differential equation*, whereas we have a *partial differential equation* when partial derivatives are present.

A differential equation (ordinary or partial) has *order* p if p is the maximum order of differentiation that is present. The next chapter will be devoted to the study of partial differential equations, whereas in the present chapter we will deal with ordinary differential equations of first order.

8.1 Some representative problems

Ordinary differential equations describe the evolution of many phenomena in various fields, as we can see from the following four examples.

Problem 8.1 (Thermodynamics) Consider a body having internal temperature T which is set in an environment with constant temperature T_e. Assume that its mass m is concentrated in a single point. Then the heat transfer between the body and the external environment can be described by the Stefan-Boltzmann law

$$v(t) = \epsilon\gamma S(T^4(t) - T_e^4),$$

where t is the time variable, ϵ the Stefan-Boltzmann constant (equal to $5.6{\cdot}10^{-8}\mathrm{J}/(\mathrm{m}^2\mathrm{K}^4\mathrm{s})$ where J stands for Joule, K for Kelvin and, obviously, m for meter, s for second), γ is the emissivity constant of the body, S the area of its surface and v is the rate of the heat transfer. The rate of variation of the energy $E(t) = mCT(t)$ (where C denotes the specific heat of the material constituting the body) equals, in absolute value,

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_8, © Springer-Verlag Berlin Heidelberg 2014

the rate v. Consequently, setting $T(0) = T_0$, the computation of $T(t)$ requires the solution of the ordinary differential equation

$$\frac{dT}{dt} = -\frac{v}{mC}. \tag{8.1}$$

See Exercise 8.15 for its solution. ■

Problem 8.2 (Population dynamics) Consider a population of bacteria in a confined environment in which no more than B elements can coexist. Assume that, at the initial time, the number of individuals is equal to $y_0 \ll B$ and the growth rate of the bacteria is a positive constant C. In this case the rate of change of the population is proportional to the number of existing bacteria, under the restriction that the total number cannot exceed B. This is expressed by the differential equation

$$\frac{dy}{dt} = Cy\left(1 - \frac{y}{B}\right), \tag{8.2}$$

whose solution $y = y(t)$ denotes the number of bacteria at time t.

Assuming that two populations y_1 and y_2 be in competition, instead of (8.2) we would have

$$\begin{aligned} \frac{dy_1}{dt} &= C_1 y_1 \left(1 - b_1 y_1 - d_2 y_2\right), \\ \frac{dy_2}{dt} &= -C_2 y_2 \left(1 - b_2 y_2 - d_1 y_1\right), \end{aligned} \tag{8.3}$$

where C_1 and C_2 represent the growth rates of the two populations. The coefficients d_1 and d_2 govern the type of interaction between the two populations, while b_1 and b_2 are related to the available quantity of nutrients. The above equations (8.3) are called the Lotka-Volterra equations and form the basis of various applications. For their numerical solution, see Example 8.7. ■

Problem 8.3 (Baseball trajectory) We want to simulate the trajectory of a ball from the pitcher to the catcher. By adopting the reference frame of Figure 8.1, the equations describing the ball motion are (see [Ada90], [GN06])

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \qquad \frac{d\mathbf{v}}{dt} = \mathbf{F},$$

where $\mathbf{x}(t) = (x(t), y(t), z(t))^T$ designates the position of the ball at time t, $\mathbf{v}(t) = (v_x(t), v_y(t), v_z(t))^T$ its velocity, while $\mathbf{F}$ is the vector whose components are

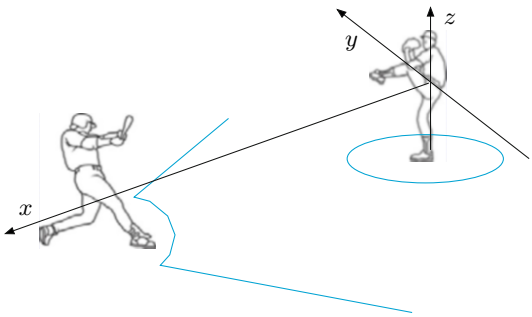

Figure 8.1. The reference frame adopted for Problem 8.3

$$\begin{aligned} F_x &= -F(v)vv_x + B\omega(v_z \sin\phi - v_y \cos\phi), \\ F_y &= -F(v)vv_y + B\omega v_x \cos\phi, \\ F_z &= -g - F(v)vv_z - B\omega v_x \sin\phi. \end{aligned} \tag{8.4}$$

v is the modulus of $\mathbf{v}$, $B = 4.1\ 10^{-4}$ a normalized constant, ϕ is the pitching angle, ω is the modulus of the angular velocity impressed to the ball from the pitcher. $F(v)$ is a friction coefficient, normally defined as ([GN06])

$$F(v) = 0.0039 + \frac{0.0058}{1 + e^{(v-35)/5}}.$$

The solution of this system of ordinary differential equations is postponed to Exercise 8.20. ■

Problem 8.4 (Electrical circuits) Consider the electrical circuit of Figure 8.2. We want to compute the function $v(t)$ representing the potential drop at the ends of the capacitor C starting from the initial time $t = 0$ at which the switch I has been turned off. Assume that the inductance L can be expressed as an explicit function of the current intensity i, that is $L = L(i)$. The Ohm law yields

$$e - \frac{d(i_1 L(i_1))}{dt} = i_1 R_1 + v,$$

where R_1 is a resistance. By assuming the current fluxes to be directed as indicated in Figure 8.2, upon differentiating with respect to t both sides of the Kirchoff law $i_1 = i_2 + i_3$ and noticing that $i_3 = C dv/dt$ and $i_2 = v/R_2$, we find the further equation

$$\frac{di_1}{dt} = C\frac{d^2v}{dt^2} + \frac{1}{R_2}\frac{dv}{dt}.$$

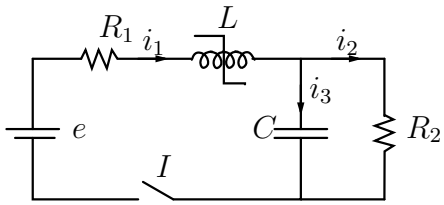

Figure 8.2. The electrical circuit of Problem 8.4

We have therefore found a system of two differential equations whose solution allows the description of the time variation of the two unknowns i_1 and v. The second equation has order two. For its solution see Example 8.8. ■

8.2 The Cauchy problem

We confine ourselves to first order differential equations, as an equation of order $p > 1$ can always be reduced to a system of p equations of order 1. The case of first order systems will be addressed in Section 8.9.

An ordinary differential equation in general admits an infinite number of solutions. In order to fix one of them we must impose a further condition which prescribes the value taken by this solution at a given point of the integration interval. For instance, the equation (8.2) admits the family of solutions $y(t) = B\psi(t)/(1 + \psi(t))$ with $\psi(t) = e^{Ct+K}$, K being an arbitrary constant. If we impose the condition $y(0) = 1$, we pick up the unique solution corresponding to the value $K = \ln[1/(B-1)]$.

We will therefore consider the solution of the so-called *Cauchy problem* which takes the following form:

find $y : I \subset \mathbb{R} \to \mathbb{R}$ such that

$$\begin{cases} y'(t) = f(t, y(t)) & \forall t \in I, \\ y(t_0) = y_0, \end{cases} \tag{8.5}$$

where I is an interval, $f : I \times \mathbb{R} \to \mathbb{R}$ is a given function, y' denotes the derivative of y with respect to t, t_0 is a point of I and y_0 a given value which is called the *initial data*.

In the following proposition we report a classical result of Analysis.

Proposition 8.1 *Assume that the function $f(t, y)$ is*

1. *continuous with respect to both its arguments;*
2. *Lipschitz-continuous with respect to its second argument, that is, there exists a positive constant L (named Lipschitz constant) such that*

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \quad \forall t \in I, \ \forall y_1, y_2 \in \mathbb{R}.$$

Then the solution $y = y(t)$ of the Cauchy problem (8.5) exists, is unique and belongs to $C^1(I)$.

Unfortunately, explicit solutions are available only for very special types of ordinary differential equations. In some other cases, the solution is available only in implicit form. This is, for instance, the case with the equation $y' = (y-t)/(y+t)$ whose solution satisfies the implicit relation

$$\frac{1}{2}\ln(t^2 + y^2) + \operatorname{arctg}\frac{y}{t} = C,$$

where C is an arbitrary constant. In some other circumstances the solution is not even representable in implicit form, as in the case of the equation $y' = e^{-t^2}$ whose general solution can only be expressed through a series expansion. For all these reasons, we seek numerical methods capable of approximating the solution of *every* family of ordinary differential equations for which solutions do exist.

The common strategy of all these methods consists of subdividing the integration interval $I = [t_0, T]$, with $T < +\infty$, into N_h intervals of length $h = (T - t_0)/N_h$; h is called the *discretization step*, or *time-step*, or *steplength*. Then, at each *node* $t_n = t_0 + nh$ (for $n = 1, \ldots, N_h$) we seek the unknown value u_n which approximates $y_n = y(t_n)$. The set of values $\{u_0 = y_0, u_1, \ldots, u_{N_h}\}$ represents our *numerical solution.*

8.3 Euler methods

A classical method, the *forward Euler* method, generates the numerical solution as follows

$$u_{n+1} = u_n + hf_n, \qquad n = 0, \ldots, N_h - 1 \tag{8.6}$$

where we have used the shorthand notation $f_n = f(t_n, u_n)$. This method is obtained by considering the differential equation (8.5) at every node t_n, $n = 1, \ldots, N_h$ and replacing the exact derivative $y'(t_n)$ by means of the incremental ratio (4.4).

In a similar way, using this time the incremental ratio (4.8) to approximate $y'(t_{n+1})$, we obtain the *backward Euler* method

$$u_{n+1} = u_n + hf_{n+1}, \qquad n = 0, \ldots, N_h - 1 \tag{8.7}$$

Both methods provide an instance of a *one-step method* since for computing the numerical solution u_{n+1} at the node t_{n+1} we only need the information related to the previous node t_n. More precisely, in the forward Euler method u_{n+1} depends exclusively on the value u_n previously computed, whereas in the backward Euler method it depends also on itself through the value f_{n+1}. For this reason the first method is called the *explicit* Euler method and the second one the *implicit* Euler method.

For instance, the discretization of (8.2) by the forward Euler method requires at every step the simple computation of

$$u_{n+1} = u_n + hCu_n\,(1 - u_n/B)\,,$$

whereas using the backward Euler method we must solve the nonlinear equation

$$u_{n+1} = u_n + hCu_{n+1}\,(1 - u_{n+1}/B)\,.$$

Thus, implicit methods are more costly than explicit methods, since, if the function f in (8.5) is not linear, at every time level t_{n+1} we must solve a nonlinear problem to compute u_{n+1}. However, we will see that implicit methods enjoy better stability properties than explicit ones.

The forward Euler method is implemented in Program 8.1; the integration interval is `tspan = [t0,tfinal]`, `odefun` is the function handle associated with the function $f(t, y(t))$ which depends on the variables t and y.

Program 8.1. feuler: forward Euler method

```
function [t,u]=feuler(odefun,tspan,y0,Nh,varargin)
%FEULER Solves differential equations using the forward
%   Euler method.
%   [T,Y]=FEULER(ODEFUN,TSPAN,Y0,NH) with TSPAN=[T0,TF]
%   integrates the system of differential equations
%   y'=f(t,y) from time T0 to TF with initial condition
%   Y0 using the forward Euler method on an equispaced
%   grid of NH intervals.
%   Function ODEFUN(T,Y) must return a vector, whose
%   elements hold the evaluation of f(t,y), of the
%   same dimension of Y.
%   Each row in the solution array Y corresponds to a
%   time returned in the  column vector T.
%   [T,Y] = FEULER(ODEFUN,TSPAN,Y0,NH,P1,P2,...) passes
%   the additional parameters P1,P2,... to the function
%   ODEFUN as ODEFUN(T,Y,P1,P2...).
```

```
h=(tspan(2)-tspan(1))/Nh;
y=y0(:); % always creates a column vector
w=y; u=y.';
tt=linspace(tspan(1),tspan(2),Nh+1);
for t = tt(1:end-1)
 w=w+h*odefun(t,w,varargin{:});
 u = [u; w.'];
end
t=tt';
return
```

The backward Euler method is implemented in Program 8.2. Note that we have used the function `fsolve` for the solution of the nonlinear problem at each step. As initial data for `fsolve` we use the last computed value of the numerical solution.

Program 8.2. beuler: backward Euler method

```
function [t,u]=beuler(odefun,tspan,y0,Nh,varargin)
%BEULER Solves differential equations using the
%  backward Euler method.
%  [T,Y]=BEULER(ODEFUN,TSPAN,Y0,NH) with TSPAN=[T0,TF]
%  integrates the system of differential equations
%  y'=f(t,y) from time T0 to TF with initial condition
%  Y0 using the backward Euler method on an equispaced
%  grid of NH intervals.
%  Function ODEFUN(T,Y) must return a vector, whose
%  elements hold the evaluation of f(t,y), of the
%  same dimension of Y.
%  Each row in the solution array Y corresponds to a
%  time returned in the  column vector T.
%  [T,Y] = BEULER(ODEFUN,TSPAN,Y0,NH,P1,P2,...) passes
%  the additional parameters P1,P2,... to the function
%  ODEFUN as ODEFUN(T,Y,P1,P2...).
tt=linspace(tspan(1),tspan(2),Nh+1);
y=y0(:); % always create a vector column
u=y.';
global glob_h glob_t glob_y glob_odefun;
glob_h=(tspan(2)-tspan(1))/Nh;
glob_y=y;
glob_odefun=odefun;
glob_t=tt(2);

if ( exist('OCTAVE_VERSION') )
o_ver=OCTAVE_VERSION;
version=str2num([o_ver(1),o_ver(3),o_ver(5)]);
end

if ( ~exist('OCTAVE_VERSION') | version >= 320 )
options=optimset;
options.Display='off';
options.TolFun=1.e-12;
options.MaxFunEvals=10000;
end
for glob_t=tt(2:end)
if ( exist('OCTAVE_VERSION') & version < 320 )
  w = fsolve('beulerfun',glob_y);
```

```
else
  w = fsolve(@(w) beulerfun(w),glob_y,options);
end
  u = [u; w.'];
  glob_y = w;
end
t=tt';
clear glob_h glob_t glob_y glob_odefun;
end

function [z]=beulerfun(w)
  global glob_h glob_t glob_y glob_odefun;
  z=w-glob_y-glob_h*glob_odefun(glob_t,w);
end
```

8.3.1 Convergence analysis

A numerical method is *convergent* if

$$\forall n = 0, \ldots, N_h, \qquad |y_n - u_n| \leq C(h) \tag{8.8}$$

where $C(h)$ is infinitesimal with respect to h when h tends to zero. If $C(h) = \mathcal{O}(h^p)$ for some $p > 0$ (that is there exists a positive constant c such that $C(h) \leq ch^p$ and p is the maximum integer for which this inequality holds), then we say that the method converges with *order* p.

In order to verify that the forward Euler method converges, we write the error as follows:

$$e_n = y_n - u_n = (y_n - u_n^*) + (u_n^* - u_n), \tag{8.9}$$

where

$$u_n^* = y_{n-1} + hf(t_{n-1}, y_{n-1})$$

denotes the numerical solution at time t_n which we would obtain starting from the exact solution at time t_{n-1}; see Figure 8.3. The term $y_n - u_n^*$ in (8.9) represents the error produced by a single step of the forward Euler method (this error is infinitesimal thanks to the consistency property), whereas the term $u_n^* - u_n$ represents the propagation from t_{n-1} to t_n of the error accumulated at the previous time level t_{n-1} (this propagation is bounded thanks to the stability property). The method converges provided both terms tend to zero as $h \to 0$; otherwise said, convergence is assured if the method is both consistent and stable.

Assuming that the second order derivative of y exists and is continuous, thanks to (4.6) we find that there exists $\xi_n \in (t_{n-1}, t_n)$ such that

$$y_n - u_n^* = \frac{h^2}{2} y''(\xi_n). \tag{8.10}$$

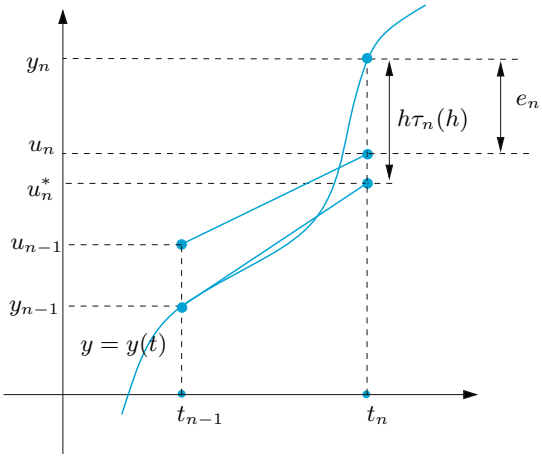

Figure 8.3. Geometrical representation of a step of the forward Euler method

The quantity

$$\tau_n(h) = (y_n - u_n^*)/h$$

is named local truncation error of the forward Euler method.

More in general, the *local truncation error* of a given method represents (up to a factor $1/h$) the error that would be generated by forcing the exact solution to satisfy that specific numerical scheme.

The *global truncation error* (or, more simply, *truncation error*) is defined as

$$\tau(h) = \max_{n=0,\dots,N_h} |\tau_n(h)|.$$

In view of (8.10), the truncation error for the forward Euler method takes the following form

$$\tau(h) = Mh/2, \tag{8.11}$$

where $M = \max_{t\in[t_0,T]} |y''(t)|$.

From (8.10) we deduce that $\lim_{h\to 0} \tau(h) = 0$, and a method for which this happens is said to be *consistent*. Further, we say that it is consistent with order p if $\tau(h) = \mathcal{O}(h^p)$ for a suitable integer $p \geq 1$.

Consider now the other term in (8.9). We have

$$u_n^* - u_n = e_{n-1} + h\left[f(t_{n-1}, y_{n-1}) - f(t_{n-1}, u_{n-1})\right]. \tag{8.12}$$

Since f is Lipschitz-continuous with respect to its second argument, we obtain

$$|u_n^* - u_n| \leq (1 + hL)|e_{n-1}|.$$

If $e_0 = 0$, the previous relations yield

$$\begin{aligned} |e_n| &\le |y_n - u_n^*| + |u_n^* - u_n| \\ &\le h|\tau_n(h)| + (1 + hL)|e_{n-1}| \\ &\le \left[1 + (1 + hL) + \ldots + (1 + hL)^{n-1}\right] h\tau(h) \\ &= \frac{(1 + hL)^n - 1}{L}\tau(h) \le \frac{e^{L(t_n - t_0)} - 1}{L}\tau(h). \end{aligned}$$

We have used the identity

$$\sum_{k=0}^{n-1}(1 + hL)^k = [(1 + hL)^n - 1]/hL,$$

the inequality $1 + hL \le e^{hL}$ and we have observed that $nh = t_n - t_0$. Therefore we find

$$|e_n| \le \frac{e^{L(t_n - t_0)} - 1}{L}\frac{M}{2}h \qquad \forall n = 0, \ldots, N_h, \tag{8.13}$$

and thus we can conclude that *the forward Euler method converges with order 1*. We can note that the order of this method coincides with the order of its local truncation error. This property is shared by many numerical methods for the numerical solution of ordinary differential equations. The convergence estimate (8.13) is now obtained by simply requiring f to be Lipschitz-continuous.

A better estimate, precisely

$$|e_n| \le Mh(t_n - t_0)/2, \tag{8.14}$$

holds if $\partial f/\partial y$ exists and satisfies the further requirement $\partial f(t, y)/\partial y \le 0$ for all $t \in [t_0, T]$ and all $-\infty < y < \infty$. Indeed, in that case, using Taylor expansion, from (8.12) we obtain

$$u_n^* - u_n = \left(1 + h\frac{\partial f}{\partial y}(t_{n-1}, \eta_n)\right) e_{n-1},$$

where η_n belongs to the interval whose endpoints are y_{n-1} and u_{n-1}, thus $|u_n^* - u_n| \le |e_{n-1}|$, provided the inequality

$$\boxed{0 < h < 2/\max_{t\in[t_0,T]}\left|\frac{\partial f}{\partial y}(t, y(t))\right|} \tag{8.15}$$

holds. Then $|e_n| \le |y_n - u_n^*| + |e_{n-1}| \le nh\tau(h) + |e_0|$, whence (8.14) owing to (8.11) and to the fact that $e_0 = 0$. The limitation (8.15) on the step h is in fact a *stability condition*, as we will see in the sequel.

Remark 8.1 (Consistency) The property of consistency is necessary in order to get convergence. Actually, should it be violated, at each step the numerical method would generate an error which is not infinitesimal with respect to h. The accumulation with the previous errors would inhibit the global error to converge to zero when $h \to 0$. ■

For the backward Euler method the local truncation error reads

$$\tau_n(h) = \frac{1}{h}[y_n - y_{n-1} - hf(t_n, y_n)].$$

Still using the Taylor expansion one obtains

$$\tau_n(h) = -\frac{h}{2}y''(\xi_n)$$

for a suitable $\xi_n \in (t_{n-1}, t_n)$, provided $y \in C^2$. Thus also the backward Euler method converges with order 1 with respect to h.

Example 8.1 Consider the Cauchy problem

$$\begin{cases} y'(t) = \cos(2y(t)), & t \in (0,1], \\ y(0) = 0, \end{cases} \tag{8.16}$$

whose solution is $y(t) = \frac{1}{2}\arcsin((e^{4t}-1)/(e^{4t}+1))$. We solve it by the forward Euler method (Program 8.1) and the backward Euler method (Program 8.2). By the following commands we use different values of h, $1/2$, $1/4, 1/8, \ldots, 1/512$:

```
tspan=[0,1]; y0=0; f=@(t,y) cos(2*y);
u=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
Nh=2;
for k=1:10
    [t,ufe]=feuler(f,tspan,y0,Nh);
    fe(k)=abs(ufe(end)-u(t(end)));
    [t,ube]=beuler(f,tspan,y0,Nh);
    be(k)=max(abs(ube-u(t)));
    Nh = 2*Nh;
end
```

The errors computed at the point $t = 1$ are stored in the variable `fe` (forward Euler) and `be` (backward Euler), respectively. Then we apply formula (1.12) to estimate the order of convergence. Using the following commands

```
p=log(abs(fe(1:end-1)./fe(2:end)))/log(2); p(1:2:end)
```

```
  1.2898    1.0349    1.0080    1.0019    1.0005


  0.8770    0.9649    0.9908    0.9978    0.9994
```

we can verify that both methods are convergent with order 1. ■

Remark 8.2 (Roundoff errors effects) The error estimate (8.13) was derived by assuming that the numerical solution $\{u_n\}$ is obtained in exact arithmetic. Should we account for the (inevitable) roundoff-errors, the error might blow up like $\mathcal{O}(1/h)$ as h approaches 0 (see, e.g., [Atk89]). This circumstance suggests that it might be unreasonable to go below a certain threshold h^* (which is actually extremely tiny) in practical computations. ■

See the Exercises 8.1-8.3.

8.4 The Crank-Nicolson method

By combining the generic steps of the forward and backward Euler methods we find the so-called *Crank-Nicolson method*

$$u_{n+1} = u_n + \frac{h}{2}[f_n + f_{n+1}], \quad n = 0, \ldots, N_h - 1 \tag{8.17}$$

This method can also be derived by applying the fundamental theorem of integration (which we recalled in Section 1.5.3) to the Cauchy problem (8.5), obtaining

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt, \tag{8.18}$$

and then approximating the integral by the trapezoidal rule (4.19).

The local truncation error of the Crank-Nicolson method satisfies

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

The last equality follows from (8.18) and expresses, up to a factor of $1/h$, the error associated with the trapezoidal rule for numerical integration (4.19). If we assume that $y \in C^3$ and use (4.20), we deduce that

$$\tau_n(h) = -\frac{h^2}{12} y'''(\xi_n) \text{ for a suitable } \xi_n \in (t_{n-1}, t_n). \tag{8.19}$$

Thus the Crank-Nicolson method is consistent with order 2, i.e. its local truncation error tends to 0 as h^2. Using a similar approach to that followed for the forward Euler method, we can show that the Crank-Nicolson method is convergent with order 2 with respect to h.

The Crank-Nicolson method is implemented in the Program 8.3. Input and output parameters are the same as for the Euler methods.

Program 8.3. cranknic: Crank-Nicolson method

```
function [t,u]=cranknic(odefun,tspan,y0,Nh,varargin)
%CRANKNIC  Solves differential equations using the
%  Crank-Nicolson method.
%  [T,Y]=CRANKNIC(ODEFUN,TSPAN,Y0,NH)  with
%  TSPAN=[T0,TF] integrates the system of differential
%  equations y'=f(t,y) from time T0 to TF with initial
%  condition Y0 using the Crank-Nicolson method on an
%  equispaced grid of NH intervals.
%  Function ODEFUN(T,Y) must return a vector, whose
%  elements hold the evaluation of f(t,y), of the
%  same dimension of Y.
%  Each row in the solution array Y corresponds to a
%  time returned in the  column vector T.
%  [T,Y] = CRANKNIC(ODEFUN,TSPAN,Y0,NH,P1,P2,...)
%  passes the additional parameters P1,P2,... to the
%  function ODEFUN as ODEFUN(T,Y,P1,P2...).
tt=linspace(tspan(1),tspan(2),Nh+1);
y=y0(:); % always create a vector column
u=y.';
global glob_h glob_t glob_y glob_odefun;
glob_h=(tspan(2)-tspan(1))/Nh;
glob_y=y;
glob_odefun=odefun;
if ( exist('OCTAVE_VERSION') )
o_ver=OCTAVE_VERSION;
version=str2num([o_ver(1),o_ver(3),o_ver(5)]);
end

if( ~exist('OCTAVE_VERSION')  | version >= 320 )
 options=optimset;
 options.Display='off';
 options.TolFun=1.e-12;
 options.MaxFunEvals=10000;
end
for glob_t=tt(2:end)
if ( exist('OCTAVE_VERSION') & version < 320 )
  w = fsolve('cranknicfun',glob_y);
else
  w = fsolve(@(w) cranknicfun(w),glob_y,options);
end
  u = [u; w.'];
  glob_y = w;
end
t=tt';
clear glob_h glob_t glob_y glob_odefun;
end

function z=cranknicfun(w)
  global glob_h glob_t glob_y glob_odefun;
  z=w - glob_y - ...
    0.5*glob_h*(glob_odefun(glob_t,w) + ...
    glob_odefun(glob_t-glob_h,glob_y));
end
```

Example 8.2 Let us solve the Cauchy problem (8.16) by using the Crank-Nicolson method with the same values of h as used in Example 8.1. The results show that the error tends to zero with order $p = 2$ with respect to h:

```
y0=0;  tspan=[0 1]; N=2; f=@(t,y) cos(2*y);
y=@(t) 0.5*asin((exp(4*t)-1)./(exp(4*t)+1));
for k=1:10
  [tt,u]=cranknic(f,tspan,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)

    1.9627    1.9986    2.0001    1.9999    2.0000
```

■

See the Exercises 8.4-8.5.

8.5 Zero-stability

Commonly speaking, by stability of a numerical scheme we mean its capability to keep the effects on the solution of data perturbations under control.

Among several concepts of stability, there is the zero-stability, which guarantees that, in a *fixed bounded interval*, small perturbations of data yield bounded perturbations of the numerical solution *when* $h \to 0$.

More precisely, a numerical method for the approximation of problem (8.5), with $I = [t_0, T]$, is *zero-stable* if

$\exists h_0 > 0$, $\exists C > 0$, $\exists \varepsilon_0 > 0$ s.t. $\forall h \in (0, h_0], \forall \varepsilon \in (0, \varepsilon_0]$, if $|\rho_n| \leq \varepsilon, 0 \leq n \leq N_h$, then

$$|z_n - u_n| \leq C\varepsilon, \qquad 0 \leq n \leq N_h, \tag{8.20}$$

where:
- C is a constant which might depend on the length $T - t_0$ of the integration interval I, but is independent of h;
- z_n is the solution that would be obtained by applying the numerical method at hand to a *perturbed* problem;
- ρ_n denotes the size of the perturbation introduced at the nth step;
- ε indicates the maximum size of the perturbation.

Obviously, ε_0 and ε must be small enough to guarantee that the perturbed problem still has a unique solution on the integration interval I.

For instance, in the case of the forward Euler method u_n satisfies the problem

$$\begin{cases} u_{n+1} = u_n + hf(t_n, u_n), & n = 0, \ldots, N_h - 1 \\ u_0 = y_0, \end{cases} \tag{8.21}$$

whereas z_n satisfies the perturbed problem

$$\begin{cases} z_{n+1} = z_n + h\left[f(t_n, z_n) + \rho_{n+1}\right], \quad n = 0, \ldots, N_h - 1 \\ z_0 = y_0 + \rho_0. \end{cases} \tag{8.22}$$

For a consistent one-step method zero-stability follows for the property of f to be Lipschitz-continuous with respect to its second argument (see, e.g. [QSS07]). In that case, the constant C that appears in (8.20) depends on $\exp((T - t_0)L)$, where L is the Lipschitz constant.

However, this is not necessarily true for other families of methods. Assume for instance that the numerical method can be written in the general form

$$u_{n+1} = \sum_{j=0}^{p} a_j u_{n-j} + h \sum_{j=0}^{p} b_j f_{n-j} + h b_{-1} f_{n+1}, \ n = p, p+1, \ldots \tag{8.23}$$

for suitable coefficients $\{a_k\}$ and $\{b_k\}$ and for an integer $p \geq 0$.

Formula (8.23) defines an important family of methods, the *linear multistep methods* and $p+1$ denotes the number of steps. These methods will be analyzed with more details in Section 8.7. The initial values $u_0, u_1, \ldots, u_p$ must be provided. Apart from u_0, which is equal to y_0, the other values $u_1, \ldots, u_p$ can be generated by suitable accurate methods such as e.g., the Runge-Kutta methods that we will address in Section 8.7.

The polynomial

$$\pi(r) = r^{p+1} - \sum_{j=0}^{p} a_j r^{p-j} \tag{8.24}$$

is called the *first characteristic polynomial* associated with the numerical method (8.23), and we denote its roots by r_j, $j = 0, \ldots, p$. It can be proved that the method (8.23) is zero-stable iff the following *root condition* is satisfied:

$$\begin{cases} |r_j| \leq 1 \text{ for all } j = 0, \ldots, p, \\ \text{furthermore } \pi'(r_j) \neq 0 \text{ for those } j \text{ such that } |r_j| = 1. \end{cases} \tag{8.25}$$

For example, for the forward Euler method we have

$$p = 0, \ a_0 = 1, \ b_{-1} = 0, \ b_0 = 1,$$

for the backward Euler method we have

$$p = 0, \ a_0 = 1, \ b_{-1} = 1, \ b_0 = 0,$$

and for the Crank-Nicolson method we have

$$p = 0,\ a_0 = 1,\ b_{-1} = 1/2,\ b_0 = 1/2.$$

In all cases there is only one root of $\pi(r)$ which is equal to 1 and therefore all these methods are zero-stable.

The following property, known as Lax-Richtmyer *equivalence theorem*, is most crucial in the theory of numerical methods (see, e.g., [IK66]), and highlights the fundamental role played by the property of zero-stability:

Any consistent method is convergent iff it is zero-stable

Coherently with what done before, the local truncation error for the multistep method (8.23) is defined as follows

$$\tau_n(h) = \frac{1}{h}\left\{ y_{n+1} - \sum_{j=0}^{p} a_j y_{n-j} - h\sum_{j=0}^{p} b_j f(t_{n-j}, y_{n-j}) - hb_{-1} f(t_{n+1}, y_{n+1}) \right\}. \tag{8.26}$$

As already noticed, the method is said to be consistent if $\tau(h) = \max|\tau_n(h)|$ tends to zero when h tends to zero. By a tedious use of Taylor expansions we can prove that this condition is equivalent to require that

$$\sum_{j=0}^{p} a_j = 1, \qquad -\sum_{j=0}^{p} j a_j + \sum_{j=-1}^{p} b_j = 1 \tag{8.27}$$

which in turns amounts to say that $r = 1$ is a root of the polynomial $\pi(r)$ introduced in (8.24) (see, e.g., [QSS07, Chapter 11]).

8.6 Stability on unbounded intervals

In the previous section we considered the solution of the Cauchy problem on bounded intervals. In that context, the number N_h of subintervals becomes infinite only if h goes to zero.

On the other hand, there are several situations in which the Cauchy problem needs to be integrated on very large (virtually infinite) time intervals. In this case, even if h is fixed, N_h tends to infinity, and then results like (8.13) become meaningless as the right hand side of the inequality contains an unbounded quantity. We are therefore interested in

methods that are able to approximate the solution for arbitrarily long time intervals, even with a steplength h relatively "large".

Unfortunately, the inexpensive forward Euler method does not enjoy this property. To see this, let us consider the following *model problem*

$$\begin{cases} y'(t) = \lambda y(t), & t \in (0, \infty), \\ y(0) = 1, \end{cases} \tag{8.28}$$

where λ is a negative real number. The exact solution is $y(t) = e^{\lambda t}$, which tends to 0 as t tends to infinity. Applying the forward Euler method to (8.28) we find that

$$u_0 = 1, \qquad u_{n+1} = u_n(1 + \lambda h) = (1 + \lambda h)^{n+1}, \qquad n \geq 0. \tag{8.29}$$

Thus $\lim_{n\to\infty} u_n = 0$ iff

$$\boxed{-1 < 1 + h\lambda < 1, \quad \text{i.e.} \quad h < 2/|\lambda|} \tag{8.30}$$

This condition expresses the requirement that, for *fixed* h, the numerical solution should reproduce the behavior of the exact solution when t_n tends to infinity. If $h > 2/|\lambda|$, then $\lim_{n\to\infty} |u_n| = +\infty$; thus (8.30) is a stability condition. The property that

$$\lim_{n\to\infty} u_n = 0 \tag{8.31}$$

is called *absolute stability*.

Example 8.3 Let us apply the forward Euler method to solve problem (8.28) with $\lambda = -1$. In that case we must have $h < 2$ for absolute stability. In Figure 8.4 we report the solutions obtained on the interval $[0, 30]$ for 3 different values of h: $h = 30/14$ (which violates the stability condition), $h = 30/16$ (which satisfies, although by a little amount only, the stability condition) and $h = 1/2$. We can see that in the first two cases the numerical solution oscillates. However only in the first case (which violates the stability condition) the absolute value of the numerical solution does not vanish at infinity (and actually it diverges). ■

Similar conclusions hold when λ is either a complex number (see Section 8.6.1) or when $\lambda = \lambda(t)$ in (8.28) is a negative function of t in (8.28). However in the latter case, $|\lambda|$ must be replaced by $\max_{t\in[0,\infty)} |\lambda(t)|$ in the stability condition (8.30). This condition could however be relaxed to one which is less restrictive by using a *variable steplength* $\overline{h}_n$ which accounts for the local behavior of $|\lambda(t)|$ in every interval (t_n, t_{n+1}).

In particular, the following *adaptive* forward Euler method could be used:

choose $u_0 = y_0$ and $\overline{h}_0 = 2\alpha/|\lambda(t_0)|$; then

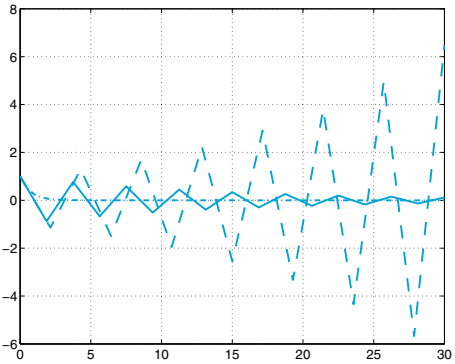

Figure 8.4. Solutions of problem (8.28), with $\lambda = -1$, obtained by the forward Euler method, corresponding to $h = 30/14$ (> 2) (*dashed line*), $h = 30/16$ (< 2) (*solid line*) and $h = 1/2$ (*dashed-dotted line*)

$$\begin{aligned} &\text{for } n = 0, 1, \ldots, \text{ do} \\ &\quad t_{n+1} = t_n + \overline{h}_n, \\ &\quad u_{n+1} = u_n + \overline{h}_n \lambda(t_n) u_n, \\ &\quad \overline{h}_{n+1} = 2\alpha / |\lambda(t_{n+1})|, \end{aligned} \tag{8.32}$$

where α is a constant which must be less than 1 in order to have an absolutely stable method.

For instance, consider the problem

$$y'(t) = -(e^{-t} + 1) y(t), \qquad t \in (0, 10),$$

with $y(0) = 1$. Since $|\lambda(t)|$ is decreasing, the most restrictive condition for absolute stability of the forward Euler method is $h < h_0 = 2/|\lambda(0)| = 1$. In Figure 8.5, left, we compare the solution of the forward Euler method with that of the adaptive method (8.32) for three values of α. Note that, although every $\alpha < 1$ is admissible for stability purposes, to get an accurate solution requires choosing α sufficiently small. In Figure 8.5, right, we also plot the behavior of $\overline{h}_n$ on the interval $(0, 10]$ corresponding to the three values of α. This picture clearly shows that the sequence $\{\overline{h}_n\}$ increases monotonically with n.

In contrast to the forward Euler method, neither the backward Euler method nor the Crank-Nicolson method require limitations on h for absolute stability. In fact, with the backward Euler method we obtain $u_{n+1} = u_n + \lambda h u_{n+1}$ and therefore

$$u_{n+1} = \left(\frac{1}{1 - \lambda h} \right)^{n+1}, \qquad n \geq 0,$$

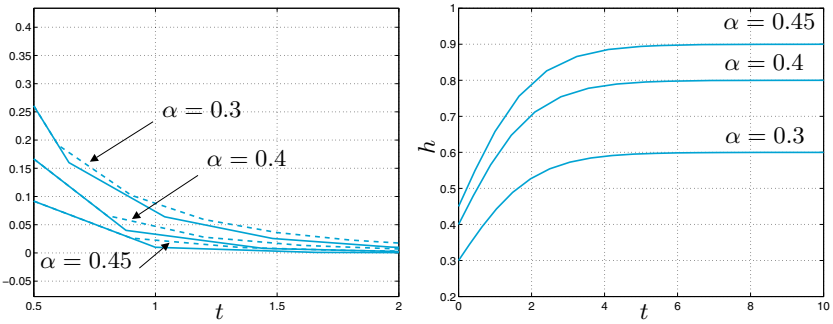

Figure 8.5. Left: the numerical solution on the time interval $(0.5, 2)$ obtained by the forward Euler method with $h = \alpha h_0$ (*dashed line*) and by the adaptive variable stepping forward Euler method (8.32) (*solid line*) for three different values of α. Right: the behavior of the variable steplength $\overline{h}_n$ for the adaptive method (8.32)

which tends to zero as $n \to \infty$ for *all values of* $h > 0$. Similarly, with the Crank-Nicolson method we obtain

$$u_{n+1} = \left[\left(1 + \frac{h\lambda}{2}\right) \Big/ \left(1 - \frac{h\lambda}{2}\right) \right]^{n+1}, \qquad n \geq 0,$$

which still tends to zero as $n \to \infty$ for all possible values of $h > 0$. We can conclude that the forward Euler method is *conditionally absolutely stable*, while both the backward Euler and Crank-Nicolson methods are *unconditionally absolutely stable*.

8.6.1 The region of absolute stability

If in (8.28) λ is a complex number with negative real part, the solution $u(t) = e^{\lambda t}$ still tends to 0 when t tends to infinity.

We call *region of absolute stability* $\mathcal{A}$ of a numerical method the set of complex numbers $z = h\lambda$ for which the method turns out to be absolutely stable (that is, $\lim_{n\to\infty} u_n = 0$).

The region of absolute stability of forward Euler method is given by those numbers $h\lambda \in \mathbb{C}$ such that $|1 + h\lambda| < 1$, thus it coincides with the circle of radius one and with centre $(-1, 0)$. This yields an upper bound $h < -2Re(\lambda)/|\lambda|^2$ for the steplength. For the backward Euler method the property of absolute stability is instead satisfied by all values of $h\lambda$ which are exterior to the circle of radius one centered in $(1, 0)$ (see Figure 8.6). Finally, the region of absolute stability of Crank-Nicolson method coincides with the left hand complex plane of numbers with negative real part.

Methods that are unconditionally absolutely stable for all complex number λ in (8.28) with negative real part are called *A-stable*. Backward

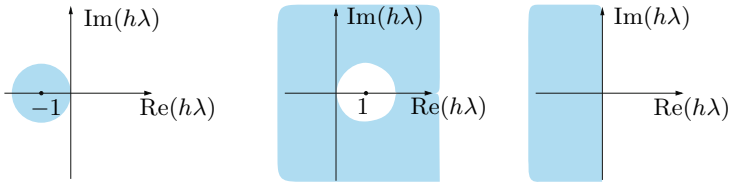

Figure 8.6. The absolute stability regions (*in cyan*) of the forward Euler method (*left*), backward Euler method (*centre*) and Crank-Nicolson method (*right*)

Euler and Crank-Nicolson method are therefore *A-stable*, and so are many other implicit methods. This property makes implicit methods attractive in spite of being computationally more expensive than explicit methods.

Example 8.4 Let us compute the restriction on h when using the forward Euler method to solve the Cauchy problem $y'(t) = \lambda y$ with $\lambda = -1 + i$. This λ stands on the boundary of the absolute stability region $\mathcal{A}$ of the forward Euler method. Thus, any h such that $h \in (0, 1)$ will suffice to guarantee that $h\lambda \in \mathcal{A}$. If it were $\lambda = -2 + 2i$ we should choose $h \in (0, 1/2)$ in order to bring $h\lambda$ within the stability region $\mathcal{A}$. ■

8.6.2 Absolute stability controls perturbations

Consider now the following *generalized model problem*

$$\begin{cases} y'(t) = \lambda(t)y(t) + r(t), & t \in (0, +\infty), \\ y(0) = 1, \end{cases} \tag{8.33}$$

where λ and r are two continuous functions and $-\lambda_{max} \leq \lambda(t) \leq -\lambda_{min}$ with $0 < \lambda_{min} \leq \lambda_{max} < +\infty$. In this case the exact solution does not necessarily tend to zero as t tends to infinity; for instance if both r and λ are constants we have

$$y(t) = \left(1 + \frac{r}{\lambda}\right) e^{\lambda t} - \frac{r}{\lambda}$$

whose limit when t tends to infinity is $-r/\lambda$. Thus, in general, it does not make sense to require a numerical method to be absolutely stable, i.e. to satisfy (8.31), when applied to problem (8.33). However, we are going to show that a numerical method which is absolutely stable on the model problem (8.28), if applied to the generalized problem (8.33), guarantees that the perturbations are kept under control as t tends to infinity (possibly under a suitable constraint on the time-step h).

For the sake of simplicity we will confine our analysis to the forward Euler method; when applied to (8.33) it reads

$$\begin{cases} u_{n+1} = u_n + h(\lambda_n u_n + r_n), \quad n \geq 0, \\ u_0 = 1 \end{cases}$$

and its solution is (see Exercise 8.9)

$$u_n = u_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} r_k \prod_{j=k+1}^{n-1} (1 + h\lambda_j), \tag{8.34}$$

where $\lambda_k = \lambda(t_k)$ and $r_k = r(t_k)$, with the convention that the last product is equal to one if $k + 1 > n - 1$. Let us consider the following "perturbed" method

$$\begin{cases} z_{n+1} = z_n + h(\lambda_n z_n + r_n + \rho_{n+1}), \quad n \geq 0, \\ z_0 = u_0 + \rho_0, \end{cases} \tag{8.35}$$

where $\rho_0, \rho_1, \ldots$ are given perturbations which are introduced at every time level. This is a simple model in which ρ_0 and ρ_{n+1}, respectively, account for the fact that neither u_0 nor r_n can be determined exactly. (Should we account for *all* roundoff errors which are actually introduced at any step, our perturbed model would be far more involved and difficult to analyze.) The solution of (8.35) reads like (8.34), provided u_k is replaced by z_k and r_k by $r_k + \rho_{k+1}$, for all $k = 0, \ldots, n-1$. Then

$$z_n - u_n = \rho_0 \prod_{k=0}^{n-1} (1 + h\lambda_k) + h \sum_{k=0}^{n-1} \rho_{k+1} \prod_{j=k+1}^{n-1} (1 + h\lambda_j). \tag{8.36}$$

The quantity $|z_n - u_n|$ is called the *perturbation error* at step n. It is worth noticing that this quantity does not depend on the function $r(t)$.

i. For the sake of exposition, let us consider first the special case where λ_k and ρ_k are two constants equal to λ and ρ, respectively. Assume that $h < h_0(\lambda) = 2/|\lambda|$, which is the condition on h that ensures the absolute stability of the forward Euler method applied to the model problem (8.28). Then, using the following identity for the geometric sum

$$\sum_{k=0}^{n-1} a^k = \frac{1 - a^n}{1 - a}, \qquad \text{if } |a| \neq 1, \tag{8.37}$$

we obtain

$$z_n - u_n = \rho \left\{ (1 + h\lambda)^n \left(1 + \frac{1}{\lambda} \right) - \frac{1}{\lambda} \right\}. \tag{8.38}$$

It follows that the perturbation error satisfies (see Exercise 8.10)

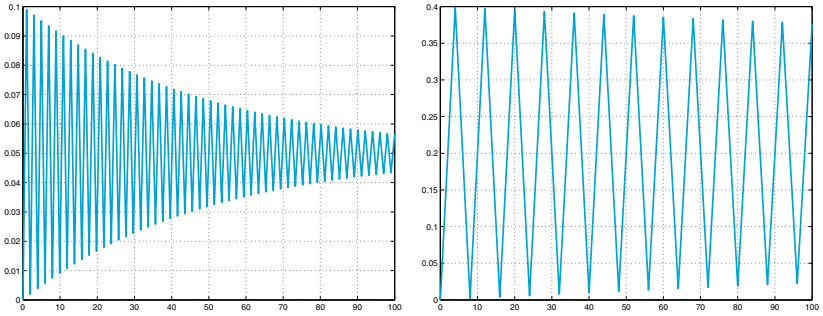

Figure 8.7. The perturbation error when $r(t) \equiv 0$, $\rho = 0.1$: $\lambda = -2$ (*left*) and $\lambda = -0.5$ (*right*). In both cases $h = h_0(\lambda) - 0.01$

$$|z_n - u_n| \leq \varphi(\lambda)|\rho|, \tag{8.39}$$

with $\varphi(\lambda) = 1$ if $\lambda \leq -1$, while $\varphi(\lambda) = |1 + 2/\lambda|$ if $-1 < \lambda < 0$. The conclusion that can be drawn is that the perturbation error is bounded by $|\rho|$ times a constant which depends on λ but is independent of both n and h. Moreover, from (8.38) it follows

$$\lim_{n\to\infty} |z_n - u_n| = \frac{|\rho|}{|\lambda|}.$$

Figure 8.7 corresponds to the case where $r(t) \equiv 0$, $\rho = 0.1$, $\lambda = -2$ (*left*) and $\lambda = -0.5$ (*right*). In both cases we have taken $h = h_0(\lambda) - 0.01$. Note that the estimate (8.38) is exactly satisfied. Obviously, the perturbation error blows up when n increases if the stability limit $h < h_0(\lambda)$ is violated.

Remark 8.3 If the unique perturbation is on the initial data, i.e. if $\rho_k = 0$, $k = 1, 2, \ldots$, from (8.36) we deduce that $\lim_{n\to\infty} |z_n - u_n| = 0$ under the stability condition $h < h_0(\lambda)$. ■

ii. In the general case where λ and r are non-constant, let us require h to satisfy the restriction $h < h_0(\lambda)$, where this time $h_0(\lambda) = 2/\lambda_{max}$. Then,

$$|1 + h\lambda_k| \leq a(h) = \max\{|1 - h\lambda_{min}|, |1 - h\lambda_{max}|\}.$$

Since $0 < \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \leq a(h) < 1$, we can still use the identity (8.37) in (8.36) and obtain

$$|z_n - u_n| \leq \overline{\rho} \left([a(h)]^n + h \frac{1 - [a(h)]^n}{1 - a(h)} \right), \tag{8.40}$$

where $\overline{\rho} = \sup_k |\rho_k|$. First, let us take $h \leq h^* = 2/(\lambda_{min} + \lambda_{max})$, so that $a(h) = (1 - h\lambda_{min})$. It holds

$$|z_n - u_n| \leq \frac{\overline{\rho}}{\lambda_{min}} \left[1 - [a(h)]^n (1 - \lambda_{min})\right], \tag{8.41}$$

i.e.,

$$\sup_n |z_n - u_n| \leq \frac{\overline{\rho}}{\lambda_{min}} \sup_n [1 - [a(h)]^n (1 - \lambda_{min})].$$

If $\lambda_{min} = 1$, we have

$$\sup_n |z_n - u_n| \leq \overline{\rho}. \tag{8.42}$$

If $\lambda_{min} < 1$, the sequence $b_n = [1 - [a(h)]^n (1 - \lambda_{min})]$ monotonically increases with n, so that $\sup_n b_n = \lim_{n\to\infty} b_n = 1$ and

$$\sup_n |z_n - u_n| \leq \frac{\overline{\rho}}{\lambda_{min}}. \tag{8.43}$$

Finally, if $\lambda_{min} > 1$, the sequence b_n monotonically decreases, $\sup_n b_n = b_0 = \lambda_{min}$, and the estimate (8.42) holds too.

Let us take now $h^* < h < h_0(\lambda)$, we have

$$1 + h\lambda_k = 1 - h|\lambda_k| \leq 1 - h^*|\lambda_k| \leq 1 - h^*\lambda_{min}. \tag{8.44}$$

Using (8.44), identity (8.37) in (8.36), and setting $a = 1 - h^*\lambda_{min}$, we find

$$\begin{aligned} z_n - u_n &\leq \overline{\rho} \left(a^n + h \frac{1 - a^n}{1 - a} \right) \\ &= \frac{\overline{\rho}}{\lambda_{min}} \left(a^n \left(\lambda_{min} - \frac{h}{h^*} \right) + \frac{h}{h^*} \right). \end{aligned} \tag{8.45}$$

We note that two possible situations arise.

If $\lambda_{min} \geq \dfrac{h}{h^*}$, then $\dfrac{h}{h^*} \leq a^n \left(\lambda_{min} - \dfrac{h}{h^*} \right) + \dfrac{h}{h^*} < \lambda_{min}$ and we find

$$z_n - u_n \leq \overline{\rho} \qquad \forall n \geq 0. \tag{8.46}$$

Otherwise, if $\lambda_{min} < \dfrac{h}{h^*}$, then $\lambda_{min} \leq a^n \left(\lambda_{min} - \frac{h}{h^*} \right) + \frac{h}{h^*} < \frac{h}{h^*}$ and

$$z_n - u_n \leq \frac{\overline{\rho}}{\lambda_{min}} \frac{h}{h^*} \leq \frac{\overline{\rho}}{\lambda_{min}} \frac{h_0}{h^*} = \overline{\rho} \left(\frac{1}{\lambda_{min}} + \frac{1}{\lambda_{max}} \right). \tag{8.47}$$

Note that the right hand side (8.47) is also an upper bound for the absolute value of $z_n - u_n$. In Figure 8.8 we report the perturbation errors computed on the problem (8.33), where $r(t) \equiv 0$, $\lambda_k = \lambda(t_k) =$

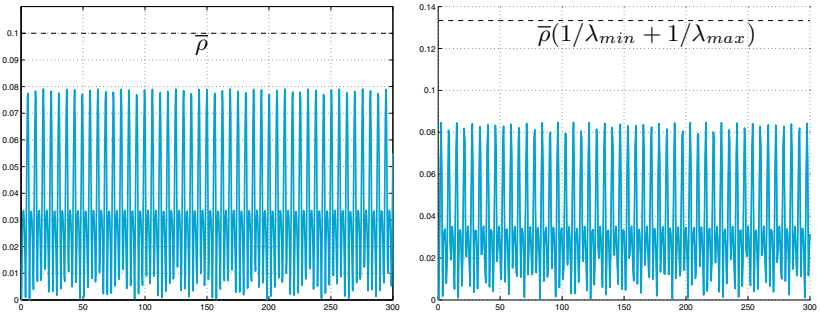

Figure 8.8. The perturbation error when $\rho(t) = 0.1\sin(t)$ and $\lambda(t) = -2 - \sin(t)$ for $t \in (0, nh)$ with $n = 500$: the steplength is $h = h^* - 0.1 = 0.4$ (*left*) and $h = h^* + 0.1 = 0.6$ (*right*). In this case $\lambda_{min} = 1$, so that the estimate (8.42) holds when $h \leq h^*$, while (8.47) holds when $h > h^*$

$-2 - \sin(t_k)$, $\rho_k = \rho(t_k) = 0.1\sin(t_k)$ with $h \leq h^*$ (*left*) and with $h^* < h < h_0(\lambda)$ (*right*).

iii. We consider now the Cauchy problem (8.5) with a general function $f(t, y(t))$. We claim that this problem can be related to the generalized model problem (8.33), in those cases where

$$-\lambda_{max} < \frac{\partial f}{\partial y}(t, y) < -\lambda_{min} \quad \forall t \geq 0,\ \forall y \in (-\infty, \infty), \tag{8.48}$$

for suitable values λ_{min}, $\lambda_{max} \in (0, +\infty)$. To this end, for every t in the generic interval (t_n, t_{n+1}), we subtract (8.6) from (8.22) to obtain the following equation for the perturbation error

$$z_n - u_n = (z_{n-1} - u_{n-1}) + h\{f(t_{n-1}, z_{n-1}) - f(t_{n-1}, u_{n-1})\} + h\rho_n.$$

By applying the mean-value theorem we obtain

$$f(t_{n-1}, z_{n-1}) - f(t_{n-1}, u_{n-1}) = \lambda_{n-1}(z_{n-1} - u_{n-1}),$$

where $\lambda_{n-1} = f_y(t_{n-1}, \xi_{n-1})$, ξ_{n-1} is a suitable point in the interval whose endpoints are u_{n-1} and z_{n-1} and f_y is a shorthand notation for $\partial f/\partial y$. Thus

$$z_n - u_n = (1 + h\lambda_{n-1})(z_{n-1} - u_{n-1}) + h\rho_n.$$

By a recursive application of this formula we obtain the identity (8.36), from which we derive the same conclusions drawn in *ii*., provided the stability restriction $0 < h < 2/\lambda_{max}$ holds. Note that this is precisely the condition (8.15).

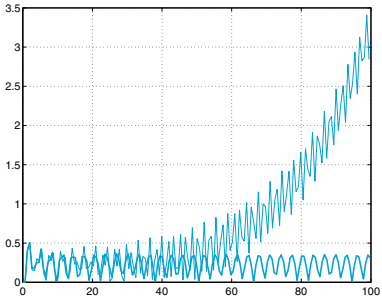

Figure 8.9. The perturbation errors when $\rho(t) = \sin(t)$ with $h = h_0 - 0.01$ (*thick line*) and $h = h_0 + 0.01$ (*thin line*) for the Cauchy problem (8.49); $h_0 = 2/3$

Example 8.5 Let us consider the Cauchy problem

$$y'(t) = \arctan(3y) - 3y + t,\ t > 0,\ y(0) = 1. \tag{8.49}$$

Since $f_y = 3/(1 + 9y^2) - 3$ is negative, we can choose $\lambda_{max} = \max|f_y| = 3$ and set $h < h_0 = 2/3$. Thus, we can expect that the perturbations on the forward Euler method are kept under control provided that $h < 2/3$. This is confirmed by the results which are reported in Figure 8.9. Note that in this example, taking $h = 2/3 + 0.01$ (thus violating the previous stability limit) the perturbation error blows up as t increases. ■

Example 8.6 We seek an upper bound on h that guarantees stability for the forward Euler method applied to approximate the Cauchy problem

$$y' = 1 - y^2, \qquad t > 0, \tag{8.50}$$

with $y(0) = \dfrac{e-1}{e+1}$. The exact solution is $y(t) = (e^{2t+1} - 1)/(e^{2t+1} + 1)$ and $f_y = -2y$. Since $f_y \in (-2, -0.9)$ for all $t > 0$, we can take h less than $h_0 = 1$. In Figure 8.10, left, we report the solutions obtained on the interval $(0, 35)$ with $h = 0.95$ (*thick line*) and $h = 1.05$ (*thin line*). In both cases the solution oscillates, but remains bounded. Moreover in the first case, which satisfies the stability constraint, the oscillations are damped and the numerical solution tends to the exact one as t increases. In Figure 8.10, right, we report the perturbation errors corresponding to $\rho(t) = \sin(t)$ with $h = h^* = 2/2.9$ (*thick solid line*) and $h = 0.9$ (*thin dashed line*). In both cases the perturbation errors remain bounded; precisely, estimate (8.42) is satisfied when $h = h^* = 2/2.9$, while estimate (8.47) holds when $h^* < h = 0.9 < h_0$. ■

In those cases where no information on y is available, finding the value $\lambda_{max} = \max|f_y|$ is not a simple matter. A more heuristic approach could be pursued in these situations, by adopting a variable stepping procedure. Precisely, one could take $t_{n+1} = t_n + h_n$, where

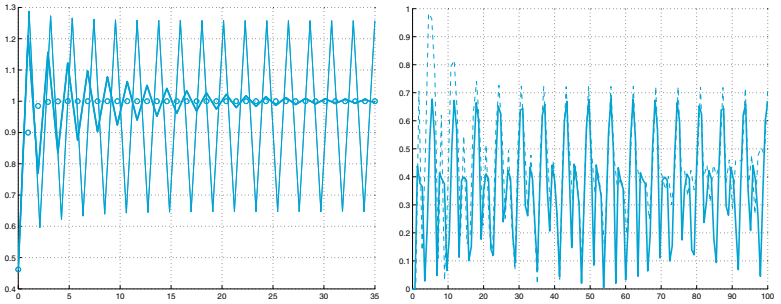

Figure 8.10. At left, numerical solutions of problem (8.50) obtained by the forward Euler method with $h = 1.05$ (*thin line*) and $h = 0.95$ (*thick line*). The values of the exact solution are indicated by circles. On the right, perturbation errors corresponding to $\rho(t) = \sin(t)$ with $h = h^* = 2/2.9$ (*thick solid line*) and $h = 0.9$ (*thin dashed line*)

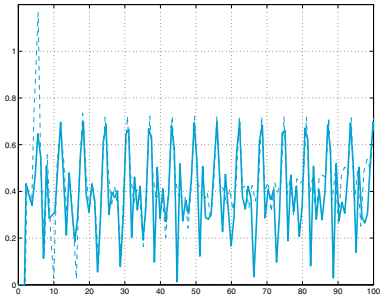

Figure 8.11. The perturbation errors corresponding to $\rho(t) = \sin(t)$ with $\alpha = 0.8$ (*thick line*) and $\alpha = 0.9$ (*thin line*) for the Example 8.6, using the adaptive strategy

$$0 < h_n < 2\frac{\alpha}{|f_y(t_n, u_n)|}, \tag{8.51}$$

for suitable values of α strictly less than 1. Note that the denominator depends on the value u_n which is known. In Figure 8.11 we report the perturbation errors corresponding to the Example 8.6 for two different values of α.

The previous analysis can be carried out also for other kind of one-step methods, in particular for the backward Euler and Crank-Nicolson methods. For these methods which are A-stable, the same conclusions about the perturbation error can be drawn without requiring any limitation on the time-step. In fact, in the previous analysis one should replace each term $1 + h\lambda_n$ by $(1 - h\lambda_n)^{-1}$ in the backward Euler case and by $(1 + h\lambda_n/2)/(1 - h\lambda_n/2)$ in the Crank-Nicolson case.

8.6.3 Stepsize adaptivity for the forward Euler method

As seen in the previous sections, the steplength h should be chosen in order to satisfy the absolute stability constraint, see e.g. (8.32) and (8.51).

More in general, at every time level we could in principle choose a (variable) time-step that not only fulfils the stability constraint but also guarantees that a desired accuracy be achieved. Such procedure, called *step adaptivity*, requires a convenient estimate of the local error, that is obtained from an appropriate a-posteriori error estimate. (A priori error estimates like (8.13) or (8.14) do not serve this porpuse, as they would require information on the second derivative of the unknown solution.) For the sake of simplicity, we illustrate this technique on the forward Euler method.

Assume that the numerical solution is computed up to a given time level that, for simplicity, will be denoted $\overline{t}$. We choose an initial guess for h and denote by u_h (respectively, $u_{h/2}$) the solution at the time $\overline{t}+h$ provided by the forward Euler method with initial value $\overline{u}$ at time $\overline{t}$ with time-step h (respectively, $h/2$), that is:

$$\begin{aligned} u_h &= \overline{u} + hf(\overline{t},\overline{u}), \\ v_1 &= \overline{u} + \frac{h}{2} f(\overline{t},\overline{u}), \end{aligned} \qquad u_{h/2} = v_2 = v_1 + \frac{h}{2} f\left(\overline{t}+\frac{h}{2}, v_1\right).$$

Let us examine the errors $e_h = y(\overline{t}+h) - u_h$ and $e_{h/2} = y(\overline{t}+h) - u_{h/2}$, where now $y(t)$ represents the exact solution to the Cauchy problem

$$\begin{cases} y'(t) = f(t, y(t)) & t \geq \overline{t}, \\ y(\overline{t}) = \overline{u}. \end{cases} \tag{8.52}$$

Using (8.10) we find

$$e_h = \frac{h^2}{2} y''(\xi) \tag{8.53}$$

for a suitable $\xi \in (\overline{t}, \overline{t}+h)$. By setting, for simplicity,

$$t_0 = \overline{t}, \qquad t_1 = \overline{t} + h/2, \qquad t_2 = \overline{t} + h$$

(see Figure 8.12) and rewriting $e_{h/2}$ in the form (8.9), we find

$$e_{h/2} = (y(t_2) - v_2^*) + (v_2^* - v_2), \tag{8.54}$$

where $v_2^* = y(t_1) + \dfrac{h}{2} f(t_1, y(t_1))$. The former term on the right hand side of (8.54) represents the local truncation error, thus

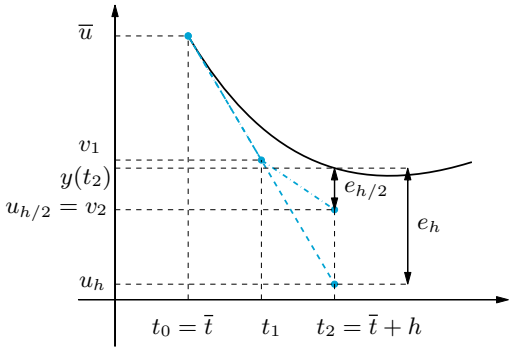

Figure 8.12. The numerical solution provided by forward Euler method with either one step of size h and two steps of size $h/2$. The solid curve represents the solution of (8.52)

$$y(t_2) - v_2^* = \frac{(h/2)^2}{2} y''(\eta_2)$$

for a suitable $\eta_2 \in (t_1, t_2)$, whereas the latter, that is due to the error propagation on an interval of length $h/2$, thanks to (8.12) reads

$$v_2^* - v_2 = (y(t_1) - v_1) + \frac{h}{2}\left[f(t_1, y(t_1)) - f(t_1, v_1)\right].$$

The term $(y(t_1) - v_1)$ still represents a local truncation error which can be written as $y(t_1) - v_1 = \dfrac{(h/2)^2}{2} y''(\eta_1)$ for a suitable $\eta_1 \in (t_0, t_1)$. On the other hand, assuming f of class C^1 and using the Lagrange theorem, we obtain

$$f(t_1, y(t_1)) = f(t_1, v_1) + (y(t_1) - v_1)\frac{\partial f}{\partial y}(t_1, \zeta)$$

for a suitable ζ belonging to the interval whose endpoints are v_1 and $y(t_1)$. Consequently

$$v_2^* - v_2 = (y(t_1) - v_1)\left[1 + h\frac{\partial f}{\partial y}(t_1, \zeta)\right] = \frac{(h/2)^2}{2} y''(\eta_1) + o(h^2).$$

Assuming that y'' is continuous in $(\bar{t}, \bar{t} + h)$, we have

$$e_{h/2} = \frac{(h/2)^2}{2}[y''(\eta_2) + y''(\eta_1)] + o(h^2) = \frac{h^2}{4} y''(\eta) + o(h^2), \qquad (8.55)$$

for a suitable $\eta \in (\bar{t}, \bar{t} + h)$.

A convenient estimate of y'' can be obtained by subtracting (8.55) from (8.53). Still assuming that y'' is continuous in $(\bar{t}, \bar{t} + h)$, we find

$$u_{h/2} - u_h = e_h - e_{h/2} = \frac{h^2}{4}\left(2y''(\xi) - y''(\eta)\right) + o(h^2) = \frac{h^2}{4}y''(\hat{\xi}) + o(h^2),$$

for a convenient $\hat{\xi} \in (\overline{t}, \overline{t} + h)$. On the other hand

$$|e_{h/2}| \simeq \frac{h^2}{4}|y''(\hat{\xi})| \simeq |u_{h/2} - u_h|,$$

therefore the quantity $|u_{h/2} - u_h|$ provides an a-posteriori estimator of the error $|y(\overline{t} + h) - u_{h/2}|$ up to an infinitesimal term $o(h^2)$.

To conclude, for a given tolerance ϵ, should

$$|u_{h/2} - u_h| < \frac{\epsilon}{2}$$

(the division by 2 is made conservatively), we accept the step h to advance and take $u_{h/2}$ as our numerical solution at the new time level $\overline{t} + h$. Otherwise, h is halved and the above procedure is repeated until convergence. In any case, to avoid too tiny steplengths we require that the steplength satisfies $h \geq h_{min}$ for a prescribed minimum value h_{min}.

We finally observe that sometimes the error estimator $|u_{h/2} - u_h|$ is replaced by its relative counterpart $|u_{h/2} - u_h|/u_{max}$, where u_{max} represents the maximum value attained by the numerical solution in the interval $[t_0, \overline{t}]$.

Let us summarize

1. An absolutely stable method is one which generates a solution u_n of the model problem (8.28) which tends to zero as t_n tends to infinity;
2. a method is said *A-stable* if it is absolutely stable for any possible choice of the time-step (or steplength) h and any $\lambda \in \mathbb{C}$ with $Re(\lambda) < 0$ (otherwise a method is called conditionally stable, and h should be lower than a constant depending on λ);
3. when an absolutely stable method is applied to a generalized model problem (like (8.33)), the perturbation error (that is the absolute value of the difference between the perturbed and unperturbed solution) is uniformly bounded with respect to h. In short, we can say that absolutely stable methods keep the perturbation controlled;
4. the analysis of absolute stability for the linear model problem can be exploited to find stability conditions on the time-step when considering the nonlinear Cauchy problem (8.5) with a function f satisfying (8.48). In that case the stability restriction requires the steplength to be chosen as a function of $\partial f/\partial y$. Precisely, the new integration interval $[t_n, t_{n+1}]$ is chosen in such a way that $h_n = t_{n+1} - t_n$ satisfies (8.51) for a suitable $\alpha \in (0, 1)$, or (8.15) in the case of constant time-step h.

See the Exercises 8.6-8.13.

8.7 High order methods

All methods presented so far are elementary examples of one-step methods. More sophisticated schemes, which allow the achievement of a higher order of accuracy, are the *Runge-Kutta methods* and the *multistep methods* (whose general form was already introduced in (7.23)).

Runge-Kutta (briefly, RK) methods are still one-step methods; however, they involve several evaluations of the function $f(t, y)$ on every interval $[t_n, t_{n+1}]$. In its most general form, a RK method can be written as

$$u_{n+1} = u_n + h\sum_{i=1}^{s} b_i K_i, \qquad n \geq 0 \tag{8.56}$$

where

$$K_i = f(t_n + c_i h, u_n + h\sum_{j=1}^{s} a_{ij} K_j), \quad i = 1, 2, \ldots, s$$

and s denotes the number of *stages* of the method. The coefficients $\{a_{ij}\}$, $\{c_i\}$ and $\{b_i\}$ fully characterize a RK method and are usually collected in the so-called *Butcher array*

$$\begin{array}{c|c} \mathbf{c} & \mathrm{A} \\ \hline & \mathbf{b}^T \end{array},$$

where $\mathrm{A} = (a_{ij}) \in \mathbb{R}^{s\times s}$, $\mathbf{b} = (b_1, \ldots, b_s)^T \in \mathbb{R}^s$ and $\mathbf{c} = (c_1, \ldots, c_s)^T \in \mathbb{R}^s$. If the coefficients a_{ij} in A are equal to zero for $j \geq i$, with $i = 1, 2, \ldots, s$, then each K_i can be explicitly computed in terms of the $i-1$ coefficients $K_1, \ldots, K_{i-1}$ that have already been determined. In such a case the RK method is *explicit*.
Otherwise, it is *implicit* and solving a nonlinear system of size s is necessary for computing the coefficients K_i.

One of the most celebrated Runge-Kutta methods reads

$$\boxed{u_{n+1} = u_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)} \tag{8.57}$$

where

$$\begin{aligned} K_1 &= f_n, \\ K_2 &= f(t_n + \tfrac{h}{2}, u_n + \tfrac{h}{2}K_1), \\ K_3 &= f(t_n + \tfrac{h}{2}, u_n + \tfrac{h}{2}K_2), \\ K_4 &= f(t_{n+1}, u_n + hK_3), \end{aligned} \qquad \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}.$$

This method can be derived from (8.18) by using the Simpson quadrature rule (4.23) to evaluate the integral between t_n and t_{n+1}. It is explicit, of fourth order with respect to h; at each time level, it involves four new evaluations of the function f. Other Runge-Kutta methods, either explicit or implicit, with arbitrary order can be constructed. For instance, an implicit RK method of order 4 with 2 stages is defined by the following Butcher array

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{3-2\sqrt{3}}{12} \\ \frac{3+\sqrt{3}}{6} & \frac{3+2\sqrt{3}}{12} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} .$$

The absolute stability region $\mathcal{A}$ of the RK methods, including explicit RK methods, can grow in surface with the order: an example is provided by the left graph in Figure 8.14, where $\mathcal{A}$ has been reported for some explicit RK methods of increasing order: RK1, i.e. the forward Euler method; RK2, the so called *improved Euler method* that will be derived later (see (8.64)); RK3, the method associated with the following Butcher array

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{8.58}$$

and RK4, the method (8.57) introduced previously.

As done for the forward Euler method, also RK method, as one-step methods, are well-suited for implementing a steplength adaptivity.
The error estimator for these methods can be constructed in two ways:
- using the same RK method, but with two different steplengths (as done for the Euler method);
- using two RK methods of different order, but with the same number s of stages.
The latter procedure is the one used by MATLAB inside the functions `ode23` and `ode45`, see below.

RK methods stand at the base of a family of MATLAB programs whose names contain the root `ode` followed by numbers and letters. In particular, `ode45` is based on a pair of explicit Runge-Kutta methods (the so-called Dormand-Prince pair) of order 4 and 5, respectively. `ode23` is the implementation of another pair of explicit Runge-Kutta methods (the Bogacki and Shampine pair). In these methods the integration step varies in order to guarantee that the error remains below a given tolerance (the default scalar relative error tolerance `RelTol` is equal to 10^{-3}). The program `ode23tb` is an implementation of an implicit Runge-Kutta

`ode` `ode45` `ode23` `ode23tb`

formula whose first stage is the trapezoidal rule, while the second stage is a backward differentiation formula of order two (see (8.61)).

Multistep methods (see (8.23)) achieve a high order of accuracy by involving the values $u_n, u_{n-1}, \ldots, u_{n-p}$ for the determination of u_{n+1}. They can be derived by applying first the formula (8.18) and then approximating the integral by a quadrature formula which involves the interpolant of f at a suitable set of nodes. A notable example of multistep method is the three-step ($p = 2$), third order (explicit) Adams-Bashforth formula (AB3)

$$u_{n+1} = u_n + \frac{h}{12}\left(23f_n - 16f_{n-1} + 5f_{n-2}\right) \tag{8.59}$$

which is obtained by replacing f in (8.18) by its interpolating polynomial of degree two at the nodes t_{n-2}, t_{n-1}, t_n. Another important example is the three-step, fourth order (implicit) Adams-Moulton formula (AM4)

$$u_{n+1} = u_n + \frac{h}{24}\left(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right) \tag{8.60}$$

which is obtained by replacing f in (8.18) by its interpolating polynomial of degree three at the nodes $t_{n-2}, t_{n-1}, t_n, t_{n+1}$.

Another family of multistep methods can be obtained by writing the differential equation at time t_{n+1} and replacing $y'(t_{n+1})$ by a one-sided incremental ratio of high order. An instance is provided by the two-step, second order (implicit) *backward difference formula* (BDF2)

$$u_{n+1} = \frac{4}{3}u_n - \frac{1}{3}u_{n-1} + \frac{2h}{3}f_{n+1} \tag{8.61}$$

or by the following three-step, third order (implicit) *backward difference formula* (BDF3)

$$u_{n+1} = \frac{18}{11}u_n - \frac{9}{11}u_{n-1} + \frac{2}{11}u_{n-2} + \frac{6h}{11}f_{n+1} \tag{8.62}$$

All these methods can be recasted in the general form (8.23). It is easy to verify that for all of them the relations (8.27) are satisfied, thus these methods are consistent. Moreover, they are zero-stable. Indeed, in both cases (8.59) and (8.60), the first characteristic polynomial is $\pi(r) = r^3 - r^2$ and its roots are $r_0 = 1$, $r_1 = r_2 = 0$; that of (8.61) is $\pi(r) = r^2 - (4/3)r + 1/3$ and its roots are $r_0 = 1$ and $r_1 = 1/3$, while the first characteristic polynomial of (8.62) is $\pi(r) = r^3 - 18/11r^2 + 9/11r - 2/11$ and its roots are $r_0 = 1$, $r_1 = 0.3182 + 0.2839i$, $r_2 = 0.3182 - 0.2839i$,

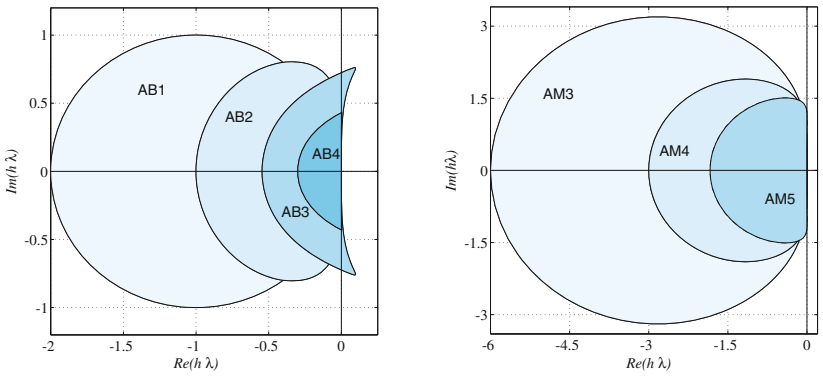

Figure 8.13. The absolute stability regions of several Adams-Basforth (*left*) and Adams-Moulton (*right*) methods

where i is the imaginary unit. In all cases, the root condition (8.25) is satisfied.

When applied to the model problem (8.28), for any $\lambda \in \mathbb{R}^-$ the method AB3 is absolutely stable if $h < 0.545/|\lambda|$, while AM4 is absolutely stable if $h < 3/|\lambda|$. The method BDF2 is unconditionally absolutely stable for any $\lambda \in \mathbb{C}$ with negative real part (i.e., A-stable). If $\lambda \in \mathbb{R}^-$, BDF3 is unconditionally absolutely stable, however this is no longer true for any $\lambda \in \mathbb{C}$ with negative real part; in other words, BDF3 fails to be A-stable (see, Figure 8.14). More generally, according to the second Dahlquist barrier there is no multistep A-stable method of order strictly greater than two.

In Figures 8.13 the regions of absolute stability of several Adams-Bashfort and Adams-Moulton methods are drawn. Note that their size reduces as far as the order increases. In the right-hand side graphs of Figure 8.14 we report the (unbounded) absolute stability regions of some BDF methods: note that $\mathcal{A}_{BDF(k+1)} \subset \mathcal{A}_{BDF(k)}$ as opposed to those of the Runge-Kutta methods (reported on the left) which instead increase in surface when the order increases, that is $\mathcal{A}_{RK(k)} \subset \mathcal{A}_{RK(k+1)}$, $k \geq 1$.

Remark 8.4 (How to draw absolute stability regions) The boundary $\partial\mathcal{A}$ of the absolute stability region $\mathcal{A}$ of a multistep method can be regarded as the set of the complex numbers $h\lambda$ such that

$$h\lambda = \left(r^{p+1} - \sum_{j=0}^{p} a_j r^{p-j} \right) \Big/ \left(\sum_{j=-1}^{p} b_j r^{p-j} \right), \tag{8.63}$$

where r is a complex number of modulus equal to one. An approximation of $\partial\mathcal{A}$ can be obtained by evaluating the right hand side of (8.63) for different values of r on the unit circle (for instance, by setting `r = exp(i*pi*(0:2000)/1000)`, where `i` is the imaginary unit). The graphs in Figures 8.13 and 8.14 have indeed been obtained in this way. ■

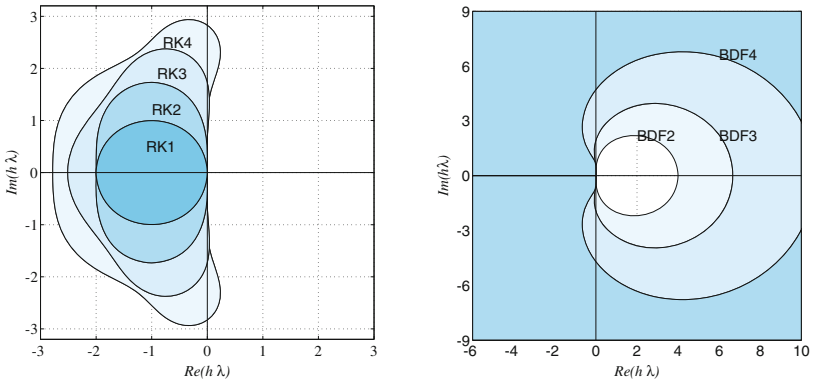

Figure 8.14. The absolute stability regions of several explicit RK (*left*) and BDF methods (*right*). In the latter case the stability regions are unbounded and they spread outside the closed curves

According to the first Dahlquist barrier the maximum order q of a $p+1$-step method satisfying the root condition is $q = p+1$ for explicit methods and, for implicit methods $q = p+2$ if $p+1$ is odd, $q = p+3$ if $p+1$ is even.

Remark 8.5 (Cyclic composite methods) Dahlquist barriers can be overcome by appropriately combining several multistep methods. For instance, the two following methods

$$u_{n+1} = -\frac{8}{11}u_n + \frac{19}{11}u_{n-1} + \frac{h}{33}(30f_{n+1} + 57f_n + 24f_{n-1} - f_{n-2}),$$
$$u_{n+1} = \frac{449}{240}u_n + \frac{19}{30}u_{n-1} - \frac{361}{240}u_{n-2} + \frac{h}{720}(251f_{n+1} + 456f_n - 1347f_{n-1} - 350f_{n-2}),$$

have order five, but are both unstable. However, combined in such a way that the former is used for n even, the latter for n odd, they give rise to an A-stable 3-step method of order five. ■

Multistep methods are implemented in several MATLAB programs, for instance in `ode15s`.

`ode15s`

Octave 8.1 `ode23` and `ode45` are also available in Octave-forge. The optional arguments however differ from MATLAB. Note that `ode45` in Octave-forge offers two possible strategies: the default one based on the Dormand and Prince method generally produces more accurate results than the other option that is based on the Fehlberg method. The built-in ODE and DAE (Differential Algebraic Equations) solvers in Octave (`lsode`, `daspk`, `dassl`, not available in MATLAB) also use multistep

methods, in particular lsode can use either Adams or BDF formulas while `dassl` and `daspk` use BDF formulas. ■

8.8 The predictor-corrector methods

In Section 8.3 it was pointed out that if the function f of Cauchy problem is nonlinear, implicit methods yield at each step a nonlinear problem for the unknown value u_{n+1}. For its solution we can use one of the methods introduced in Chapter 2, or else apply the function `fsolve` as we have done with the Programs 8.2 and 8.3.

Alternatively, we can carry out fixed point iterations at every time level. For example, for the Crank-Nicolson method (8.17), for $k = 0, 1, \ldots$, we compute until convergence

$$u_{n+1}^{(k+1)} = u_n + \frac{h}{2}\left[f_n + f(t_{n+1}, u_{n+1}^{(k)})\right].$$

It can be proved that if the initial guess $u_{n+1}^{(0)}$ is chosen conveniently, a single iteration suffices in order to obtain a numerical solution $u_{n+1}^{(1)}$ whose accuracy is of the same order as the solution u_{n+1} of the original implicit method. More precisely, if the original implicit method has order $p \geq 2$, then the initial guess $u_{n+1}^{(0)}$ must be generated by an explicit method of order (at least) $p - 1$.

For instance, if we use the first order (explicit) forward Euler method to initialize the Crank-Nicolson method, we get the *Heun method* (also called *improved Euler method*), already referred as RK2:

$$\begin{aligned} u_{n+1}^* &= u_n + h f_n, \\ u_{n+1} &= u_n + \frac{h}{2}\left[f_n + f(t_{n+1}, u_{n+1}^*)\right] \end{aligned} \tag{8.64}$$

The explicit step is called a *predictor*, whereas the implicit one is called a *corrector*. Another example combines the (AB3) method (8.59) as predictor with the (AM4) method (8.60) as corrector. These kinds of methods are therefore called *predictor-corrector* methods. They enjoy the order of accuracy of the corrector method. However, being explicit, they undergo a stability restriction which is typically the same as that of the predictor method (see, for instance, the regions of absolute stability of Figure 8.15). Thus they are not adequate to integrate a Cauchy problem on unbounded intervals.

In Program 8.4 we implement a general predictor-corrector method. The function handles `predictor` and `corrector` identify the type of method that is chosen. For instance, if we use the functions `feonestep` and `cnonestep`, which are defined in Programs 8.5 and 8.7, respectively, we can call `predcor` as follows

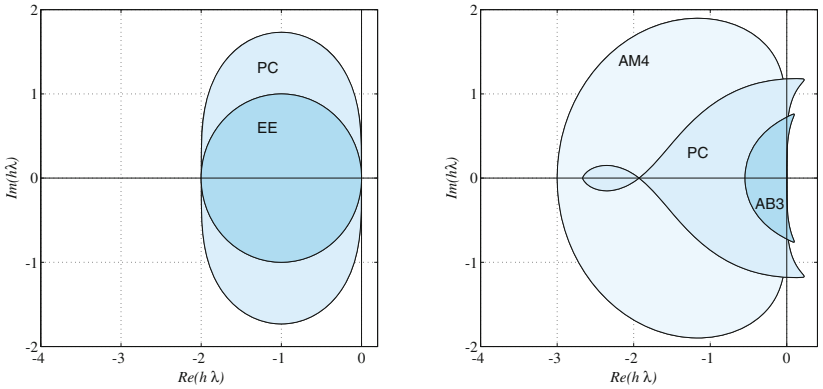

Figure 8.15. The absolute stability regions of the predictor-corrector (PC) methods obtained by combining the explicit Euler (EE) and Crank-Nicolson methods (*left*) and AB3 and AM4 (*right*). Notice the reduced surface of the region when compared to the corresponding implicit methods (in the first case the region of the Crank-Nicolson method hasn't been reported as it coincides with all the complex half-plane $Re(h\lambda) < 0$)

```
[t,u]=predcor(f,[t0,T],y0,N,@feonestep,@cnonestep);
```

and obtain the Heun method.

Program 8.4. predcor: predictor-corrector method

```
function [t,u]=predcor(odefun,tspan,y0,Nh,...
                predictor,corrector,varargin)
%PREDCOR  Solves differential equations using a
%  predictor-corrector method
%  [T,Y]=PREDCOR(ODEFUN,TSPAN,Y0,NH,PRED,CORR) with
%  TSPAN=[T0 TF] integrates the system of differential
%  equations y' = f(t,y) from time T0 to TF with
%  initial condition Y0 using a general predictor
%  corrector method on an equispaced grid of NH steps.
%  Function ODEFUN(T,Y) must return a vector, whose
%  elements hold the evaluation of f(t,y), of the
%  same dimension of Y.
%  Each row in the solution array Y corresponds to a
%  time returned in the  column vector T.
%  [T,Y]=PREDCOR(ODEFUN,TSPAN,Y0,NH,PRED,CORR,P1,..)
%  passes the additional parameters P1,... to the
%  functions ODEFUN,PRED and CORR as ODEFUN(T,Y,P1,..),
%  PRED(T,Y,P1,P2...), CORR(T,Y,P1,P2...).
h=(tspan(2)-tspan(1))/Nh;
y=y0(:); w=y; u=y.';
tt=linspace(tspan(1),tspan(2),Nh+1);
for t=tt(1:end-1)
   fn = odefun(t,w,varargin{:});
   upre = predictor(t,w,h,fn);
   w = corrector(t+h,w,upre,h,odefun,...
          fn,varargin{:});
```

```
    u = [u; w.'];
end
t = tt';
end
```

Program 8.5. feonestep: one step of the forward Euler method

```
function [u]=feonestep(t,y,h,f)
% FEONESTEP one step of the forward Euler method
u = y + h*f;
return
```

Program 8.6. beonestep: one step of the backward Euler method

```
function [u]=beonestep(t,u,y,h,f,fn,varargin)
% BEONESTEP  one step of the backward Euler method
u = u + h*f(t,y,varargin{:});
return
```

Program 8.7. cnonestep: one step of the Crank-Nicolson method

```
function [u]=cnonestep(t,u,y,h,f,fn,varargin)
% CNONESTEP one step of the Crank-Nicolson method
u = u + 0.5*h*(f(t,y,varargin{:})+fn);
return
```

The MATLAB program `ode113` implements a combined Adams-Bashforth-Moulton scheme with variable steplength. `ode113`

See the Exercises 8.14-8.17.

8.9 Systems of differential equations

Let us consider the following system of first-order ordinary differential equations whose unknowns are $y_1(t), \ldots, y_m(t)$:

$$\begin{cases} y_1' = f_1(t, y_1, \ldots, y_m), \\ \vdots \\ y_m' = f_m(t, y_1, \ldots, y_m), \end{cases}$$

where $t \in (t_0, T]$, with the initial conditions

$$y_1(t_0) = y_{0,1}, \ \ldots, \ y_m(t_0) = y_{0,m}.$$

For its solution we could apply to each individual equation one of the methods previously introduced for a scalar problem. For instance, the nth step of the forward Euler method would read

$$\begin{cases} u_{n+1,1} = u_{n,1} + hf_1(t_n, u_{n,1}, \ldots, u_{n,m}), \\ \vdots \\ u_{n+1,m} = u_{n,m} + hf_m(t_n, u_{n,1}, \ldots, u_{n,m}). \end{cases}$$

By writing the system in vector form $\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t))$, with obvious choice of notation, the extension of the methods previously developed for the case of a single equation to the vector case is straightforward. For instance, the method

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h(\vartheta \mathbf{F}(t_{n+1}, \mathbf{u}_{n+1}) + (1-\vartheta)\mathbf{F}(t_n, \mathbf{u}_n)), \qquad n \geq 0,$$

with $\mathbf{u}_0 = \mathbf{y}_0$, $0 \leq \vartheta \leq 1$, is the vector form of the forward Euler method if $\vartheta = 0$, the backward Euler method if $\vartheta = 1$ and the Crank-Nicolson method if $\vartheta = 1/2$.

Example 8.7 (Population dynamics) Let us apply the forward Euler method to solve the Lotka-Volterra equations (8.3) with $C_1 = C_2 = 1$, $b_1 = b_2 = 0$ and $d_1 = d_2 = 1$. In order to use Program 8.1 for a *system* of ordinary differential equations, let us create a function `f` which contains the component of the vector function $\mathbf{F}$, which we save in the file `f.m`. For our specific system we have:

```
function fn = f(t,y,C1,C2,d1,d2,b1,b2)
[n,m]=size(y); fn=zeros(n,m);
fn(1)=C1*y(1)*(1-b1*y(1)-d2*y(2));
fn(2)=-C2*y(2)*(1-b2*y(2)-d1*y(1));
return
```

Now we execute Program 8.1 with the following instructions

```
C1=1; C2=1; d1=1; d2=1; b1=0; b2=0;
[t,u]=feuler(@f,[0,10],[2 2],20000,C1,C2,d1,d2,b1,b2);
```

They correspond to solving the Lotka-Volterra system on the time interval $[0, 10]$ with a time-step $h = 5 \cdot 10^{-4}$.

The graph in Figure 8.16, left, represents the time evolution of the two components of the solution. Note that they are periodic. The graph in Figure 8.16, right, shows the trajectory issuing from the initial value in the so-called *phase plane*, that is, the Cartesian plane whose coordinate axes are y_1 and y_2. This trajectory is confined within a bounded region of the (y_1, y_2) plane. If we start from the point $(1.2, 1.2)$, the trajectory would stay in an even smaller region surrounding the point $(1, 1)$. This can be explained as follows. Our differential system admits 2 *points of equilibrium* at which $y_1' = 0$ and $y_2' = 0$, and one of them is precisely $(1, 1)$ (the other being $(0, 0)$). Actually, they are obtained by solving the nonlinear system

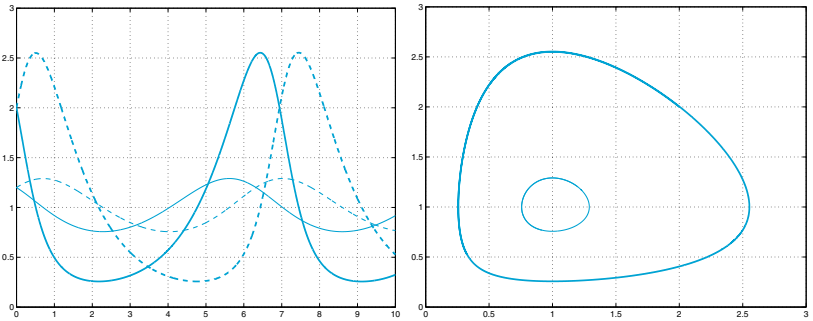

Figure 8.16. Numerical solutions of system (8.3). At left, we represent y_1 and y_2 on the time interval $(0, 10)$, the solid line refers to y_1, the dashed line to y_2. Two different initial data are considered: $(2, 2)$ (*thick lines*) and $(1.2, 1.2)$ (*thin lines*). At right, we report the corresponding trajectories in the phase plane

$$\begin{cases} y_1' = y_1 - y_1 y_2 = 0, \\ y_2' = -y_2 + y_2 y_1 = 0. \end{cases}$$

If the initial data coincide with one of these points, the solution remains constant in time. Moreover, while $(0, 0)$ is an unstable equilibrium point, $(1, 1)$ is stable, that is, all trajectories issuing from a point near $(1, 1)$ stay bounded in the phase plane. ■

When we use an explicit method, the steplength h should undergo a stability restriction similar to the one encountered in Section 8.6. When the real part of the eigenvalues λ_k of the Jacobian $\mathrm{A}(t) = [\partial \mathbf{F}/\partial \mathbf{y}](t, \mathbf{y})$ of $\mathbf{F}$ are all negative, we can set $\lambda = -\max_t \rho(\mathrm{A}(t))$, where $\rho(\mathrm{A}(t))$ is the spectral radius of $\mathrm{A}(t)$. This λ is a candidate to replace the one entering in the stability conditions (such as, e.g., (8.30)) that were derived for the scalar Cauchy problem.

Remark 8.6 The MATLAB programs (`ode23`, `ode45`, ...) that we have mentioned before can be used also for the solution of systems of ordinary differential equations. The syntax is `odeXX(@f,[t0 tf],y0)`, where `y0` is the vector of the initial conditions, `f` is a function to be specified by the user and `odeXX` is one of the methods available in MATLAB. ■

Now consider the case of an ordinary differential equation of order m

$$y^{(m)}(t) = f(t, y, y', \ldots, y^{(m-1)}) \tag{8.65}$$

for $t \in (t_0, T]$, whose solution (when existing) is a family of functions defined up to m arbitrary constants. The latter can be fixed by prescribing m initial conditions

$$y(t_0) = y_0,\ y'(t_0) = y_1,\ \ldots,\ y^{(m-1)}(t_0) = y_{m-1}.$$

Setting

$$w_1(t) = y(t),\ w_2(t) = y'(t),\ \ldots,\ w_m(t) = y^{(m-1)}(t),$$

the equation (8.65) can be transformed into a first-order system of m differential equations

$$\begin{cases} w_1' = w_2, \\ w_2' = w_3, \\ \vdots \\ w_{m-1}' = w_m, \\ w_m' = f(t, w_1, \ldots, w_m), \end{cases}$$

with initial conditions

$$w_1(t_0) = y_0,\ w_2(t_0) = y_1,\ \ldots,\ w_m(t_0) = y_{m-1}.$$

Thus we can always approximate the solution of a differential equation of order $m > 1$ by resorting to the equivalent system of m first-order equations, and then applying to this system a convenient discretization method.

Example 8.8 (Electrical circuits) Consider the circuit of Problem 8.4 and suppose that $L(i_1) = L$ is constant and that $R_1 = R_2 = R$. In this case v can be obtained by solving the following system of two differential equations:

$$\begin{cases} v'(t) = w(t), \\ w'(t) = -\dfrac{1}{LC}\left(\dfrac{L}{R} + RC\right) w(t) - \dfrac{2}{LC} v(t) + \dfrac{e}{LC}, \end{cases} \tag{8.66}$$

with initial conditions $v(0) = 0$, $w(0) = 0$. The system has been obtained from the second-order differential equation

$$LC\frac{d^2v}{dt^2} + \left(\frac{L}{R_2} + R_1C\right)\frac{dv}{dt} + \left(\frac{R_1}{R_2} + 1\right) v = e. \tag{8.67}$$

We set $L = 0.1$ Henry, $C = 10^{-3}$ Farad, $R = 10$ Ohm and $e = 5$ Volt, where Henry, Farad, Ohm and Volt are respectively the unit measure of inductance, capacitance, resistance and voltage. Now we apply the forward Euler method with $h = 0.001$ seconds in the time interval $[0, 0.1]$, by the Program 8.1:

```
L=0.1; C=1.e-03; R=10; e=5;
[t,u]=feuler(@fsys,[0,0.1],[0 0],100,L,C,R,e);
```

where `fsys` is contained in the file `fsys.m`:

```
function fn=fsys(t,y,L,C,R,e)
LC = L*C;
[n,m]=size(y); fn=zeros(n,m);
fn(1)=y(2);
fn(2)=-(L/R+R*C)/(LC)*y(2)-2/(LC)*y(1)+e/(LC);
return
```

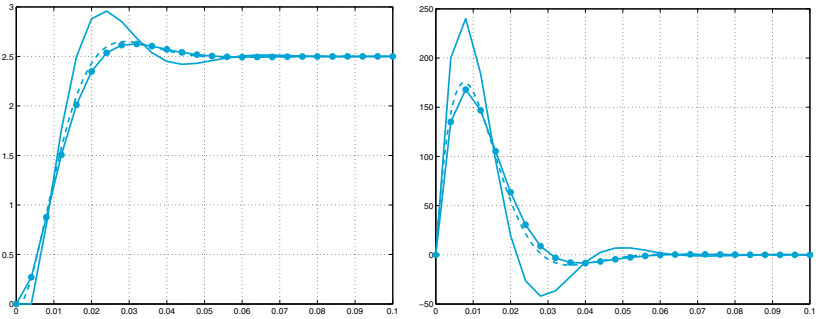

Figure 8.17. Numerical solutions of system (8.66). The potential drop $v(t)$ is reported on the left, its derivative $w(t)$ on the right: the dashed line represents the solution obtained for $h = 0.001$ with the forward Euler method, the solid line is for the one generated via the same method with $h = 0.004$, and the solid line with circles is for the one produced via the Newmark method (8.71) (with $\zeta = 1/4$ and $\theta = 1/2$) with $h = 0.004$

In Figure 8.17 we report the approximated values of $v(t)$ and $w(t)$. As expected, $v(t)$ tends to $e/2 = 2.5$ Volt for $t \to \infty$. In this case, the matrix $\mathrm{A} = [\partial \mathbf{F}/\partial \mathbf{y}](t, \mathbf{y}) = [0, 1; -20000, -200]$, hence does not depend on time. Its eigenvalues are $\lambda_{1,2} = -100 \pm 100i$, so that the bound on time-step which guarantees absolute stability is $h < -2Re(\lambda_i)/|\lambda_i|^2 = 0.01$. ■

Sometimes numerical approximations can be directly derived on the high order equation without passing through the equivalent first order system. Consider for instance the case of the 2nd order Cauchy problem

$$\begin{cases} y''(t) = f(t, y(t), y'(t)) & t \in (t_0, T], \\ y(t_0) = \alpha_0, \quad y'(t_0) = \beta_0. \end{cases} \tag{8.68}$$

Two sequences u_n and v_n will approximate $y(t_n)$ and $y'(t_n)$, respectively. A simple numerical scheme can be constructed as follows: find u_{n+1} such that

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(t_n, u_n, v_n), \qquad n = 1, \ldots, N_h, \tag{8.69}$$

with $u_0 = \alpha_0$ and $v_0 = \beta_0$. Moreover, since $(y_{n+1} - 2y_n + y_{n-1})/h^2$ is a second order approximation of $y''(t_n)$, let us consider a second order approximation for $y'(t_n)$ too, i.e. (see (4.9))

$$v_n = \frac{u_{n+1} - u_{n-1}}{2h}, \text{ with } v_0 = \beta_0. \tag{8.70}$$

The *leap-frog method* (8.69)-(8.70) is accurate of order 2 with respect to h.

A more general method is the *Newmark method*, in which we build two sequences with same meaning as before

$$\begin{aligned} u_{n+1} &= u_n + hv_n + h^2\left[\zeta f(t_{n+1}, u_{n+1}, v_{n+1})\right. \\ &\quad \left. +(1/2-\zeta) f(t_n, u_n, v_n)\right], \\ v_{n+1} &= v_n + h\left[(1-\theta) f(t_n, u_n, v_n) + \theta f(t_{n+1}, u_{n+1}, v_{n+1})\right], \end{aligned} \tag{8.71}$$

with $u_0 = \alpha_0$ and $v_0 = \beta_0$, and ζ and θ are two non-negative real numbers. This method is implicit unless $\zeta = \theta = 0$, second order if $\theta = 1/2$, whereas it is first order accurate if $\theta \neq 1/2$. The condition $\theta \geq 1/2$ is necessary to ensure stability. For $\theta = 1/2$ and $\zeta = 1/4$ we find a rather popular method that is unconditionally stable. However, this method is not suitable for simulations on long time intervals as it introduces oscillatory spurious solutions. For these simulations it is preferable to use $\theta > 1/2$ and $\zeta > (\theta + 1/2)^2/4$ even though the method degenerates to a first order one.

In Program 8.8 we implement the Newmark method. The vector `param` allows to specify the values of the coefficients (`param(1)`=ζ, `param(2)`=θ).

Program 8.8. newmark: Newmark method

```
function [t,u]=newmark(odefun,tspan,y0,Nh,param,...
                varargin)
%NEWMARK Solves second order differential equations
%  using the Newmark method
%  [T,Y]=NEWMARK(ODEFUN,TSPAN,Y0,NH,PARAM) with TSPAN =
%  [T0 TF] integrates the system of differential
%  equations y''=f(t,y,y') from time T0 to TF with
%  initial conditions Y0=(y(t0),y'(t0)) using the
%  Newmark method on an equispaced grid of NH steps.
%  PARAM holds parameters zeta and theta
%  Function ODEFUN(T,Y) must return a vector, whose
%  elements hold the evaluation of f(t,y), of the
%  same dimension of Y.
%  Each row in the solution array Y corresponds to a
%  time returned in the  column vector T.
tt=linspace(tspan(1),tspan(2),Nh+1);
y=y0(:); u=y.';
global glob_h glob_t glob_y glob_odefun;
global glob_zeta glob_theta glob_varargin glob_fn;
glob_h=(tspan(2)-tspan(1))/Nh;
glob_y=y; glob_odefun=odefun;
glob_zeta = param(1); glob_theta = param(2);
glob_varargin=varargin;
if ( exist('OCTAVE_VERSION') )
o_ver=OCTAVE_VERSION;
version=str2num([o_ver(1),o_ver(3),o_ver(5)]);
end

if ( ~exist( 'OCTAVE_VERSION' )  | version >= 320 )
 options=optimset;
```

```
 options.Display='off';
 options.TolFun=1.e-12;
 options.MaxFunEvals=10000;
end
glob_fn =odefun(tt(1),glob_y,varargin{:});
for glob_t=tt(2:end)
if ( exist( 'OCTAVE_VERSION' ) & version < 320 )
  w = fsolve('newmarkfun', glob_y );
else
  w = fsolve(@(w) newmarkfun(w),glob_y,options);
end
  glob_fn =odefun(glob_t,w,varargin{:});
  u = [u; w.']; glob_y = w;
end
t=tt';
clear glob_h glob_t glob_y glob_odefun;
clear glob_zeta glob_theta glob_varargin glob_fn;
end

function z=newmarkfun(w)
 global glob_h glob_t glob_y glob_odefun;
 global glob_zeta glob_theta glob_varargin glob_fn;
 fn1=glob_odefun(glob_t,w,glob_varargin{:});
 z(1)=w(1) - glob_y(1) -glob_h*glob_y(2)-...
   glob_h^2*(glob_zeta*fn1+(0.5-glob_zeta)*glob_fn);
 z(2)=w(2) - glob_y(2) -...
   glob_h*((1-glob_theta)*glob_fn+glob_theta*fn1);
end
```

Example 8.9 (Electrical circuits) We consider again the circuit of Problem 8.4 and we solve the second order equation (8.67) with the Newmark scheme. In Figure 8.17 we compare the numerical approximations of the function v computed using the forward Euler scheme (*dashed line* for $h = 0.001$ and *continuous line* for $h = 0.004$) and the Newmark scheme with $\theta = 1/2$ and $\zeta = 1/4$ (*solid line with circles*), with the time-step $h = 0.004$. The better accuracy of the latter solution is due to the fact that the method (8.71) is second order accurate with respect to h. ■

See the Exercises 8.18-8.20.

8.10 Some examples

We end this chapter by considering and solving three non-trivial examples of systems of ordinary differential equations.

8.10.1 The spherical pendulum

The motion of a point $\mathbf{x}(t) = (x_1(t), x_2(t), x_3(t))^T$ with mass m subject to the gravity force $\mathbf{F} = (0, 0, -gm)^T$ (with $g = 9.8$ m/s^2) and constrained to move on the spherical surface of equation $\Phi(\mathbf{x}) =$

$x_1^2 + x_2^2 + x_3^2 - 1 = 0$ is described by the following system of ordinary differential equations

$$\ddot{\mathbf{x}} = \frac{1}{m}\left(\mathbf{F} - \frac{m\,\dot{\mathbf{x}}^T\,\mathrm{H}\,\dot{\mathbf{x}} + \nabla\Phi^T\mathbf{F}}{|\nabla\Phi|^2}\nabla\Phi\right) \quad \text{for } t > 0. \tag{8.72}$$

We denote by $\dot{\mathbf{x}}$ the first derivative with respect to t, with $\ddot{\mathbf{x}}$ the second derivative, with $\nabla\Phi$ the spatial gradient of Φ, equal to $2\mathbf{x}$, with H the Hessian matrix of Φ whose components are $\mathrm{H}_{ij} = \partial^2\Phi/\partial x_i\partial x_j$ for $i, j = 1, 2, 3$. In our case H is a diagonal matrix with coefficients all equal to 2. System (8.72) must be provided with the initial conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $\dot{\mathbf{x}}\,(0) = \mathbf{v}_0$.

To numerically solve (8.72) let us transform it into a system of differential equations of order 1 in the new variable $\mathbf{y}$, a vector with 6 components. Having set $y_i = x_i$ and $y_{i+3} = \dot{x_i}$ with $i = 1, 2, 3$, and

$$\lambda = \frac{m(y_4, y_5, y_6)^T\mathrm{H}(y_4, y_5, y_6) + \nabla\Phi^T\mathbf{F}}{|\nabla\Phi|^2},$$

we obtain, for $i = 1, 2, 3$,

$$\begin{aligned} \dot{y_i} &= y_{3+i}, \\ \dot{y}_{3+i} &= \frac{1}{m}\left(F_i - \lambda\frac{\partial\Phi}{\partial y_i}\right). \end{aligned} \tag{8.73}$$

We apply the Euler and Crank-Nicolson methods. Initially it is necessary to define a MATLAB *function* (`fvinc` in Program 8.9) which yields the expressions of the right-hand terms (8.73). Furthermore, let us suppose that the initial conditions are given by vector `y0=[0,1,0,.8,0,1.2]` and that the integration interval is `tspan=[0,25]`. We recall the explicit Euler method in the following way

```
[t,y]=feuler(@fvinc,tspan,y0,nt);
```

(the backward Euler `beuler` and Crank-Nicolson `cranknic` methods can be called in the same way), where `nt` is the number of intervals (of constant width) used to discretize the interval `[tspan(1),tspan(2)]`. In the graphs in Figure 8.18 we report the trajectories obtained with 10000 and 100000 discretization nodes. Only in the second case, the solution looks reasonably accurate. As a matter of fact, although we do not know the exact solution to the problem, we can have an idea of the accuracy by noticing that the solution satisfies $r(\mathbf{y}) \equiv |y_1^2 + y_2^2 + y_3^2 - 1| = 0$ and by consequently measuring the maximal value of the residual $r(\mathbf{y}_n)$ when n varies, $\mathbf{y}_n$ being the approximation of the exact solution generated at time t_n. By using 10000 discretization nodes we find $r = 1.0578$, while with 100000 nodes we have $r = 0.1111$, in accordance with the theory requiring the explicit Euler method to converge with order 1.

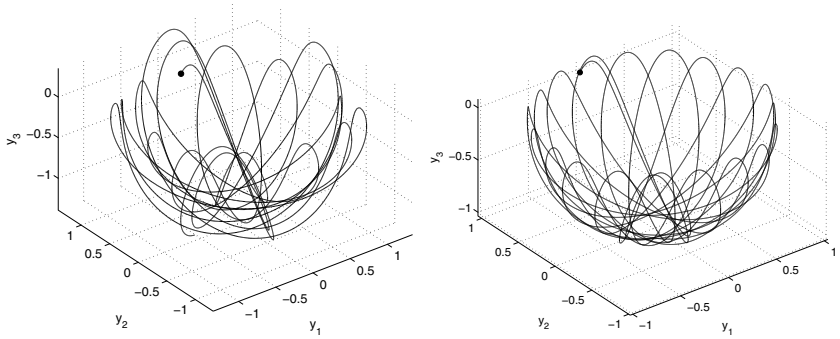

Figure 8.18. The trajectories obtained with the explicit Euler method with $h = 0.0025$ (*on the left*) and $h = 0.00025$ (*on the right*). The blackened point shows the initial datum

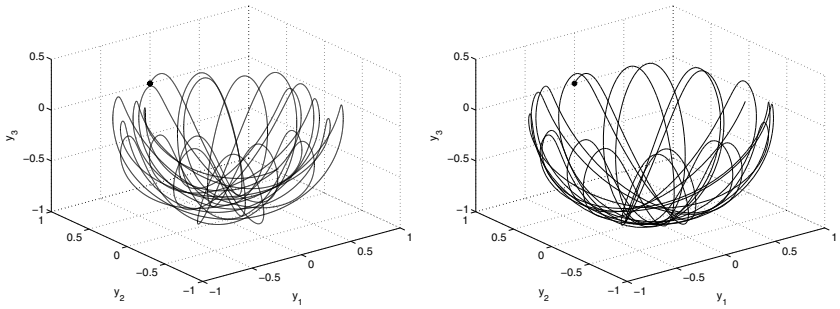

Figure 8.19. The trajectories obtained using the implicit Euler method with $h = 0.00125$ (*on the left*) and using the Crank-Nicolson method with $h = 0.025$ (*on the right*)

By using the implicit Euler method with 20000 steps we obtain the solution reported in Figure 8.19, while the Crank-Nicolson method (of order 2) with only 1000 steps provides the solution reported in the same figure on the right, which is undoubtedly more accurate. Indeed, we find $r = 0.5816$ for the implicit Euler method and $r = 0.0928$ for the Crank-Nicolson method.

As a comparison, let us solve the same problem using the explicit adaptive methods of type Runge-Kutta `ode23` and `ode45`, featured in MATLAB. These (unless differently specified) modify the integration step in order to guarantee that the relative error on the solution is less than 10^{-3} and the absolute error is less than 10^{-6}. We run them using the following commands

```
[t1,y1]=ode23(@fvinc,tspan,y0);
[t2,y2]=ode45(@fvinc,tspan,y0);
```

obtaining the solutions in Figure 8.20.

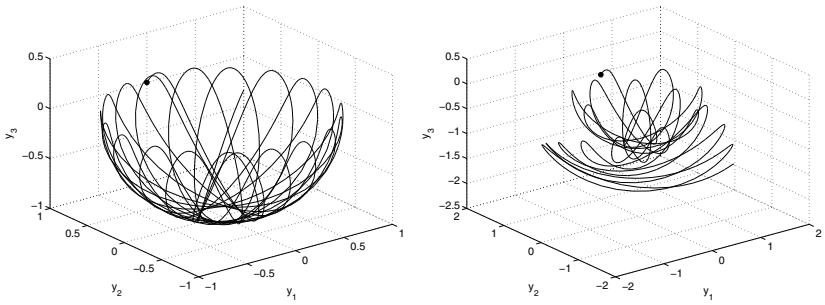

Figure 8.20. The trajectories obtained using methods `ode23` (*left*) and `ode45` (*right*) with the same accuracy criteria. In the second case the error control fails and the solution obtained is less accurate

The two methods used 783, respectively 537, non-uniformly distributed discretization nodes. The residual r is equal to 0.0238 for `ode23` and 3.2563 for `ode45`. Surprisingly, the result obtained with the highest-order method is thus less accurate and this warns us as to using the `ode` programs available in MATLAB. An explanation of this behavior is in the fact that the error estimator implemented in `ode45` is less constraining than that in `ode23`. By slightly decreasing the relative tolerance (it is sufficient to set `options=odeset('RelTol',1.e-04)`) and renaming the program to `[t,y]=ode45(@fvinc,tspan,y0,options);` we can in fact find results comparable with those of `ode23`. Precisely `ode23` requires 1751 discretization nodes and it provides a residual $r = 0.003$, while `ode45` requires 1089 discretization nodes and it provides a residual $r = 0.060$.

Program 8.9. fvinc: forcing term for the spherical pendulum problem

```
function [f]=fvinc(t,y)
[n,m]=size(y); f=zeros(n,m);
phix=2*y(1); phiy=2*y(2); phiz=2*y(3);
H=2*eye(3);
mass=1;
F1=0; F2=0; F3=-mass*9.8;
xp=zeros(3,1);
xp(1:3)=y(4:6);
F=[F1;F2;F3];
G=[phix;phiy;phiz];
lambda=(mass*xp'*H*xp+F'*G)/(G'*G);
f(1:3)=y(4:6);
for k=1:3;
  f(k+3)=(F(k)-lambda*G(k))/mass;
end
return
```

Octave 8.2 `ode23` requires 924 steps while `ode45` requires 575 steps for the same accuracy `tol=1.e-03`.

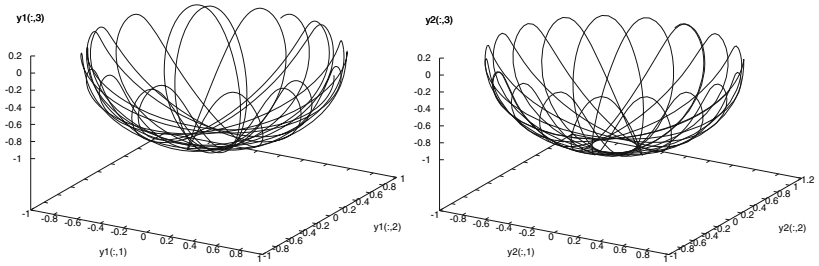

Figure 8.21. The trajectories obtained using methods `ode23` (*left*) and `ode45` (*right*) with the same accuracy criteria.

Note that `ode45` gives results similar to `ode23` as opposed to `ode45` in MATLAB, see Figure 8.21. ■

8.10.2 The three-body problem

We want to compute the evolution of a system composed by three bodies, knowing their initial positions and velocities and their masses under the influence of their reciprocal gravitational attraction. The problem can be formulated by using Newton's laws of motion. However, as opposed to the case of two bodies, there are no known closed form solutions. We suppose that one of the three bodies has considerably larger mass than the two remaining, and in particular we study the case of the Sun-Earth-Mars system, a problem studied by celeber mathematicians such as Lagrange in the eighteenth century, Poincaré towards the end of the nineteenth century and Levi-Civita in the twentieth century.

We denote by M_s the mass of the Sun, by M_e that of the Earth and by M_m that of Mars. The Sun's mass being about 330000 times that of the Earth and the mass of Mars being about one tenth of the Earth's, we can imagine that the center of gravity of the three bodies approximately coincides with the center of the Sun (which will therefore remain still in this model) and that the three objects remain in the plane described by their initial positions. In such case the total force exerted on the Earth will be for instance

$$\mathbf{F}_e = \mathbf{F}_{es} + \mathbf{F}_{em} = M_e \frac{d^2 \mathbf{x}_e}{dt^2}, \tag{8.74}$$

where $\mathbf{x}_e = (x_e, y_e)^T$ denotes the Earth's position with respect to the Sun, while $\mathbf{F}_{es}$ and $\mathbf{F}_{em}$ denote the force exerted by the Sun and by Mars, respectively, on the Earth. By applying the universal gravitational law, denoting by G the universal gravity constant and by $\mathbf{x}_m$ the position of Mars with respect to the Sun, equation (8.74) becomes

$$M_e \frac{d^2\mathbf{x}_e}{dt^2} = -GM_eM_s\frac{\mathbf{x}_e}{|\mathbf{x}_e|^3} + GM_eM_m\frac{\mathbf{x}_m - \mathbf{x}_e}{|\mathbf{x}_m - \mathbf{x}_e|^3}.$$

Now, let us take the astronomical unit (1AU) as unit length, the year (1yr) as temporal unit and define the Sun mass as $M_s = \frac{4\pi^2(1\text{AU})^3}{G(1\text{yr})^2}$. By adimensionalizing the previous equations and denoting again with $\mathbf{x}_e$, $\mathbf{x}_m$, $\mathbf{x}_s$ and t the adimensionalized variables, we obtain the following equation

$$\frac{d^2\mathbf{x}_e}{dt^2} = 4\pi^2\left(\frac{M_m}{M_s}\frac{\mathbf{x}_m - \mathbf{x}_e}{|\mathbf{x}_m - \mathbf{x}_e|^3} - \frac{\mathbf{x}_e}{|\mathbf{x}_e|^3}\right). \tag{8.75}$$

A similar equation for planet Mars can be obtained using a similar computation

$$\frac{d^2\mathbf{x}_m}{dt^2} = 4\pi^2\left(\frac{M_e}{M_s}\frac{\mathbf{x}_e - \mathbf{x}_m}{|\mathbf{x}_e - \mathbf{x}_m|^3} - \frac{\mathbf{x}_m}{|\mathbf{x}_m|^3}\right). \tag{8.76}$$

The second-order system (8.75)-(8.76) immediately reduces to a system of eight equations of order one. Program 8.10 allows to evaluate a *function* containing the right-hand side terms of system (8.75)-(8.76).

Program 8.10. threebody: forcing term for the simplified three body system

```
function f=threebody(t,y)
[n,m]=size(y); f=zeros(n,m); Ms=330000; Me=1; Mm=0.1;
D1 = ((y(5)-y(1))^2+(y(7)-y(3))^2)^(3/2);
D2 = (y(1)^2+y(3)^2)^(3/2);
f(1)=y(2); f(2)=4*pi^2*(Me/Ms*(y(5)-y(1))/D1-y(1)/D2);
f(3)=y(4); f(4)=4*pi^2*(Me/Ms*(y(7)-y(3))/D1-y(3)/D2);
D2 = (y(5)^2+y(7)^2)^(3/2);
f(5)=y(6); f(6)=4*pi^2*(Mm/Ms*(y(1)-y(5))/D1-y(5)/D2);
f(7)=y(8); f(8)=4*pi^2*(Mm/Ms*(y(3)-y(7))/D1-y(7)/D2);
return
```

Let us compare the implicit Crank-Nicolson method and the explicit adaptive Runge-Kutta method implemented in `ode23`. Having set the Earth to be 1 unit away from the Sun, Mars will be located at about 1.52 units: the initial position will therefore be $(1, 0)$ for the Earth and $(1.52, 0)$ for Mars. Let us further suppose that the two planets initially have null horizontal velocity and vertical velocity equal to -5.1 units (Earth) and -4.6 units (Mars): this way they should move along reasonably stable orbits around the Sun. For the Crank-Nicolson method we choose 2000 discretization steps:

```
[t23,u23]=ode23(@threebody,[0 10],...
                [1.52 0 0 -4.6 1 0 0 -5.1]);
[tcn,ucn]=cranknic(@threebody,[0 10],...
                   [1.52 0 0 -4.6 1 0 0 -5.1],2000);
```

The graphs in Figure 8.22 show that the two methods are both able to reproduce the elliptical orbits of the two planets around the Sun. Method

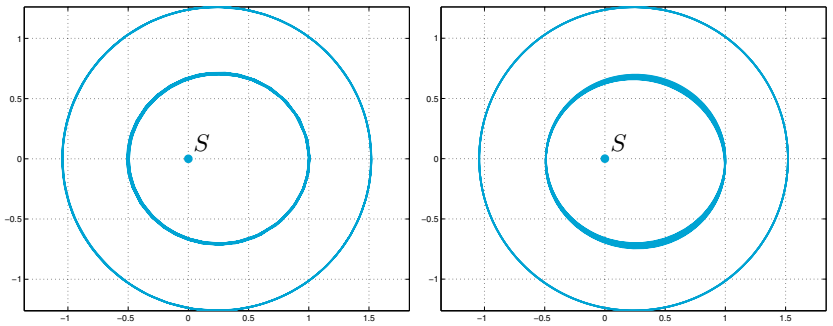

Figure 8.22. The Earth's (*inmost*) and Mars's orbit with respect to the Sun as computed with the adaptive method `ode23` (*on the left*) (with 543 steps) and with the Crank-Nicolson method (*on the right*) (with 2000 steps)

`ode23` only required 543 (nonuniform) steps to generate a more accurate solution than that generated by an implicit method with the same order of accuracy, but which does not use step adaptivity.

Octave 8.3 `ode23` requires 847 steps to generate a solution with a tolerance of 1e-3. ■

8.10.3 Some stiff problems

Let us consider the following differential problem, proposed by [Gea71], as a variant of the model problem (8.28):

$$\begin{cases} y'(t) = \lambda(y(t) - g(t)) + g'(t), & t > 0, \\ y(0) = y_0, \end{cases} \tag{8.77}$$

where g is a regular function and $\lambda < 0$ has a very large absolute value, whose solution

$$y(t) = (y_0 - g(0))e^{\lambda t} + g(t), \qquad t \geq 0. \tag{8.78}$$

is the sum of two components, also called *transient* and *persistent* solution, respectively. Initially, on a time interval of length $\mathcal{O}(1/\lambda)$, the transient component prevails, whereas the persistent component becomes predominant in the asymptotic regime (for sufficiently large t).

In particular, we set $g(t) = t$, $\lambda = -100$, and $y_0 = 1$ and solve problem (8.77) over the interval $(0, 100)$ using the explicit Euler method: since in this case $f(t, y) = \lambda(y(t) - g(t)) + g'(t)$ we have $\partial f/\partial y = \lambda$, and the stability analysis performed in Section 8.5 suggests that we choose $h < 2/100$. This restriction is dictated by the presence of the component behaving like e^{-100t} and appears completely unjustified when we think

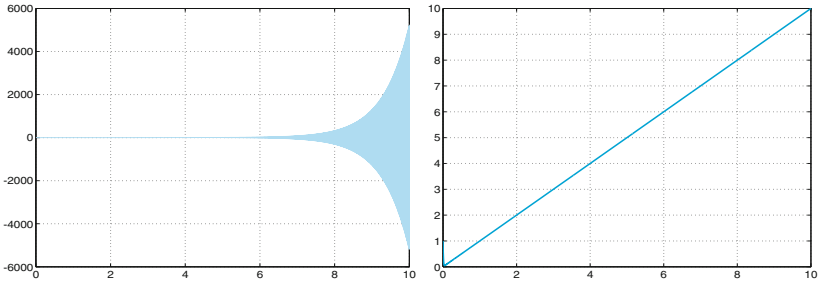

Figure 8.23. Solutions obtained using method (8.59) for problem (8.77) violating the stability condition ($h = 0.0055$, *left*) and respecting it ($h = 0.0054$, *right*)

of its weight with respect to the whole solution for sufficiently large t (to get an idea, for $t = 1$ we have $e^{-100} \approx 10^{-44}$). The situation gets worse using a higher order explicit method, such as for instance the Adams-Bashforth (8.59) method of order 3: the absolute stability region reduces (see Figure 8.13) and, consequently, the restriction on h becomes even stricter, $h < 0.00545$. Violating – even slightly – such restriction produces unacceptable solutions (as shown in Figure 8.23 on the left).

We thus face an apparently simple problem, but one that becomes difficult to solve with an explicit method (and more generally with a method which is not A-stable).

In fact, even though for large values of t it is the persistent component of the solution that prevails (in the current case it is a straight line), yet for its correct approximation we must enforce a strong limitation on h. Such kind of problem is called *stiff*, or, more precisely, it is a stiff problem on the interval on which the persistent solution prevails. As a matter of fact the choice of h is subjected to stability constraints; in these cases, the use of explicit methods, even if implemented using adaptive strategies, is prohibitive.

Programs implementing adaptive methods do not explicitely check that absolute stability condition is satisfied. Nevertheless, the error estimator provides steplength h such that $h\lambda$ belongs to the absolute stability region.

We consider now a system of linear differential equations that reads

$$\mathbf{y}'(t) = \mathrm{A}\mathbf{y}(t) + \boldsymbol{\varphi}(t), \qquad \mathrm{A} \in \mathbb{R}^{n\times n}, \quad \boldsymbol{\varphi}(t) \in \mathbb{R}^n, \tag{8.79}$$

where A has n distinct eigenvalues λ_j, $j = 1, \ldots, n$ with $\mathrm{Re}(\lambda_j) < 0$. Its exact solution is

$$\mathbf{y}(t) = \sum_{j=1}^{n} C_j e^{\lambda_j t}\mathbf{v}_j + \boldsymbol{\psi}(t), \tag{8.80}$$

where $C_1, \ldots, C_n$ are n costants and $\{\mathbf{v}_j\}$ is a basis of $\mathbb{R}^n$ whose components are the eigenvectors of A, while $\boldsymbol{\psi}(t)$ is a special solution of (8.79).

Similarly to the scalar case (8.78), $C_j e^{\lambda_j t}\mathbf{v}_j$ represent the transient components of the solution and $\boldsymbol{\psi}(t)$ the persistent component (for large t). If $|\mathrm{Re}(\lambda_j)|$ is large, the corresponding transient component will tend to zero very quickly, while for small values of $|\mathrm{Re}(\lambda_j)|$, the corresponding transient components will decay more slowly. If we approximate (8.79) by a numerical scheme that is not absolutely stable, the transient component featuring the largest value of $|\mathrm{Re}(\lambda_j)|$ is the one that yields the most stringent constraint on the steplength h, even though such component is the quickest to decay to zero.

A parameter that is often used to measure the stiff character of a system is

$$r_s = \frac{\max_j |\mathrm{Re}(\lambda_j)|}{\min_j |\mathrm{Re}(\lambda_j)|},$$

even though by itself r_s is not fully meaningful. As a matter of fact, the stiff character of a system depends on r_s, the eigenvalues of A, the initial conditions, the persistent component of the solution and the time interval on which the system has to be solved.

On the other hand, the stiff character depends not only on the form of the exact solution of (8.79); as a matter of fact there exist different systems, some of them stiff, some other non-stiff, all featuring the same exact solution, see, e.g., [Lam91, Ch. 6].
How can we therefore state whether a system is stiff or not? Let us quote the following definition proposed by [Lam91, pag. 220].

Definition 8.1 *A system of ordinary differential equations is said* stiff *if, once approximated by a numerical method featuring an absolute stability region of bounded extension, "forces" the said numerical method, for every initial condition for which the given problem admits a solution, to use a steplength exceedingly small with respect to the one that would be necessary to reasonably reproduce the behavior of the exact solution.*

In the case of problem (8.77) (or (8.79)) the system is not stiff in the initial interval where the solution varies quickly, whence the need of adopting a small h to well capture the sharp layer. Rather, it is stiff in the next interval where the solution features a mild slope. Within this interval the fastest transient, although exhausted because negligible with respect to the other components, still dictates the choice of a tiny steplength h because of stability constraints.

A-stable numerical methods (those whose absolute stability region comprises the half complex plane $Re\lambda < 0$) with adaptive choice of the steplength are the most efficient for *stiff* problems. Their implicit character makes them more computationally involved than explicit methods, however they can afford much larger steplengths. Explicit methods, on their turn, may be unaffordable because of the strong limitation on h.

The algorithm implemented in function `ode15s` is based on multistep methods and backward differentiation formulas BDF introduced in Section 8.7. Its formal convergence order is variable and at most 5. This method is very effective also on systems that are non-stiff for which the Jacobian matrix of $\mathbf{f}(t, \mathbf{y})$ is either constant or features very small variations.

`ode23s` The function `ode23s` implements a linear implicit multistep method based on Rosenbrock methods see [SR97] for a detailed description of these two functions.

Example 8.10 Let us consider the system $\mathbf{y}'(t) = \mathrm{A}\mathbf{y}(t)$, $t \in (0, 100)$ with initial condition $\mathbf{y}(0) = \mathbf{y}_0$, where $\mathbf{y} = (y_1, y_2)^T$, $\mathbf{y}_0 = (y_{1,0}, y_{2,0})^T$ and

$$\mathrm{A} = \begin{bmatrix} 0 & 1 \\ -\lambda_1\lambda_2 & \lambda_1 + \lambda_2 \end{bmatrix},$$

where λ_1 and λ_2 are two different negative numbers such that $|\lambda_1| \gg |\lambda_2|$. Matrix A has eigenvalues λ_1 and λ_2 and eigenvectors $\mathbf{v}_1 = (1, \lambda_1)^T$, $\mathbf{v}_2 = (1, \lambda_2)^T$. Thanks to (8.80) the exact solution is

$$\mathbf{y}(t) = \begin{pmatrix} C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} \\ C_1\lambda_1 e^{\lambda_1 t} + C_2\lambda_2 e^{\lambda_2 t} \end{pmatrix}^T. \tag{8.81}$$

The constants C_1 and C_2 are obtained by fulfilling the initial condition:

$$C_1 = \frac{\lambda_2 y_{1,0} - y_{2,0}}{\lambda_2 - \lambda_1}, \qquad C_2 = \frac{y_{2,0} - \lambda_1 y_{1,0}}{\lambda_2 - \lambda_1}.$$

Based on the remarks made earlier, the integration step of an explicit method used for the resolution of such a system will depend uniquely on the eigenvalue having largest modulus, λ_1. Let us assess this experimentally using the explicit Euler method and choosing $\lambda_1 = -100$, $\lambda_2 = -1$ (therefore $r_s = 100$), $y_{1,0} = y_{2,0} = 1$. In Figure 8.24 we report the solutions computed by violating (*left*) or respecting (*right*) the stability condition $h < 1/50$. ■

The definition of stiff problem can be extended, with some care, to the nonlinear case (see for instance [QSS07, Chapter 11]). One of the most studied nonlinear *stiff* problems is given by the *Van der Pol equation*

$$\frac{d^2x}{dt^2} = \mu(1 - x^2)\frac{dx}{dt} - x, \tag{8.82}$$

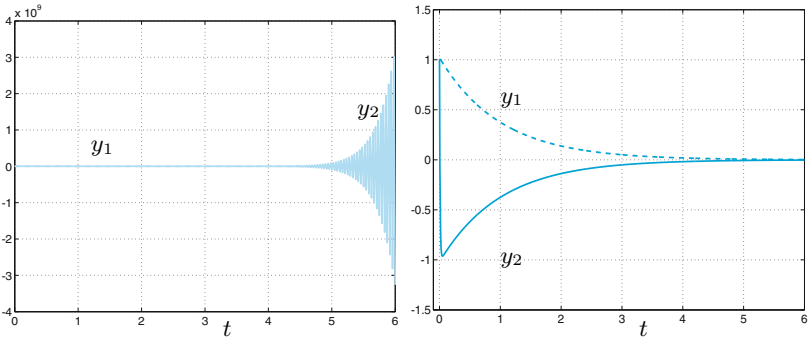

Figure 8.24. Solutions to the problem in Example 8.10 for $h = 0.0207$ (*left*) and $h = 0.01$ (*right*). In the first case the condition $h < 2/|\lambda_1| = 0.02$ is violated and the method is unstable. The second case features a strong variation of the fast transient component y_2. Consider the totally different scale in the two graphs

proposed in 1920 and used in the study of circuits containing thermionic valves, the so-called vacuum tubes, such as cathodic tubes in television sets or magnetrons in microwave ovens.

If we set $\mathbf{y} = (x, z)^T$, with $z = dx/dt$, (8.82) is equivalent to the following nonlinear first order system

$$\mathbf{y}' = \mathbf{F}(t, \mathbf{y}) = \begin{bmatrix} z \\ -x + \mu(1 - x^2)z \end{bmatrix}. \tag{8.83}$$

Such system becomes increasingly stiff with the increase of the μ parameter. In the solution we find in fact two components which denote totally different dynamics with the increase of μ. The one having the fastest dynamics imposes a limitation on the integration step which gets more and more prohibitive with the increase of μ.

If we solve (8.82) using `ode23` and `ode45`, we realize that these are too costly when μ is large. With $\mu = 100$ and initial condition $\mathbf{y} = (1, 1)^T$, `ode23` requires 7835 steps and `ode45` 23473 steps to integrate between $t = 0$ and $t = 100$. Reading the MATLAB *help* we discover that these methods are based on explicit schemes and therefore they are not recommended for stiff problems: for these, other procedures are suggested, such as for instance the implicit methods `ode23s` or `ode15s`. The difference in terms of number of steps is remarkable, as shown in Table 8.1. Notice however that the number of steps for `ode23s` is smaller than that for `ode23` only for large enough values of μ (thus for very stiff problems).

`ode23s`

Example 8.11 (Chemical kinetics) We want to investigate the temporal behavior of chemical reactions of species in homogeneous media. Quite often,

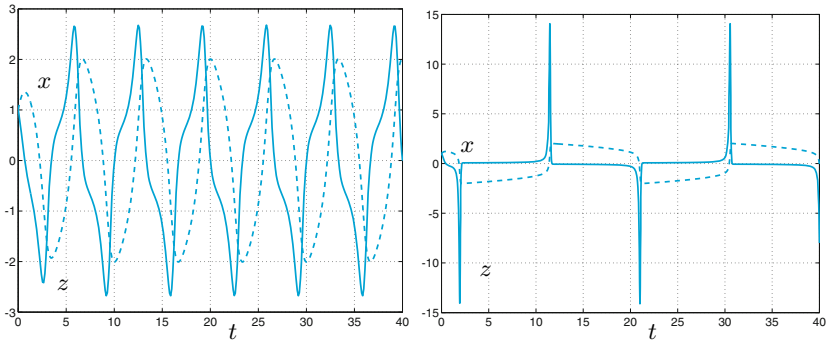

Figure 8.25. Behavior of the components of the solutions $\mathbf{y}$ to system (8.83) for $\mu = 1$ (*left*) and $\mu = 10$ (*right*)

Table 8.1. Behavior of the number of integration steps for various approximation methods with growing μ parameter

μ	`ode23`	`ode45`	`ode23s`	`ode15s`
0.1	471	509	614	586
1	775	1065	838	975
10	1220	2809	1005	1077
100	7835	23473	299	305
1000	112823	342265	183	220

both fast and slow species cohexist, that evolve according to differente characteristic times. Below we consider a mathematical model that represents a simplified version of this process. This model, named Davis-Skodje (see, e.g., [VGCN05]), addresses two species $y_1(t)$ and $y_2(t)$ that evolve according to the equations

$$\begin{cases} \dfrac{dy_1}{dt} = \dfrac{1}{\varepsilon}\left(-y_1 + \dfrac{y_2}{1+y_2}\right) - \dfrac{y_2}{(1+y_2)^2}, & t > 0 \\ \dfrac{dy_2}{dt} = -y_2, & t > 0 \\ y_1(0) = y_{1,0} \\ y_2(0) = y_{2,0}, \end{cases} \tag{8.84}$$

where $\varepsilon > 0$, $y_{1,0}$ and $y_{2,0}$ are given.

The exact solution is:

$$\begin{aligned} y_1(t) &= \left(y_{1,0} - \frac{y_{2,0}}{1+y_{2,0}}\right) e^{-t/\varepsilon} + \frac{y_{2,0}e^{-t}}{1+y_{2,0}e^{-t}} \\ y_2(t) &= y_{2,0}e^{-t}. \end{aligned}$$

The ratio $1/\varepsilon$ is a measure of the system's stiffness: the larger $1/\varepsilon$ the wider the gap between the temporal scales of the evolution of the two species, than the more complex is the numerical computation.

To numerically solve system (8.84) with $\varepsilon = 10^{-6}$ and initial condition $\mathbf{y}_0 = (1.5, 1)^T$, we have defined the function

Table 8.2. Number of steps used by a few MATLAB functions to solve problem (8.84) for different values of the parameter ε

ε	ode23	ode45	ode23s	ode15s
10^{-2}	409	1241	73	73
10^{-3}	3991	12081	84	81
10^{-4}	39808	120553	87	85

```
function [f]=funds(t,y)
epsilon=1.e-6; [n,m]=size(y);f=zeros(n,m);
f(1)=-1/epsilon*y(1)+( (1/epsilon-1)*y(2)+...
      1/epsilon*y(2)*y(2))/(1+y(2))^2;
f(2)=-y(2);
end
```

Then we call the MATLAB function `ode23s` by the following commands

```
y0=[1.5,1]; tspan=[0,10];
[t,y]=ode23s(@funds,tspan,y0);
```

In Table 8.2 we report the number of steps required by the explicit methods `ode23`, `ode45`, and by the implicit methods `ode24s`, `ode15s`. We can appreciate the better efficiency of methods `ode23s` and `ode15s`, as they have been specifically designed for stiff equations.

In Figure 8.26, left, we plot numerical solutions: the species y_1 evolves very quickly at the beginning of the simulation during a time interval of length $\mathcal{O}(\varepsilon)$, and very slowly after. On the contrary, the species y_2 varies slowly and uniformly during the whole simulation time.

In Figure 8.26, right, trajectories of problem (8.84) are shown for $\varepsilon = 10^{-6}$ and with several initial conditions $[y_{1,0}, y_{2,0}]^T$. Horizontal stretches of trajectories are covered in a very short initial time interval of length $\mathcal{O}(\varepsilon)$, while the curved ways are covered in the remaining time of length $10 - \mathcal{O}(\varepsilon)$. Analysis of trajectories can be useful to acquire carachteristic information of the chemical process. ■

Octave 8.4 While `ode15s` and `ode23s` are not available in Octave, many ODE solvers capable of dealing with stiff problems are available in the Octave core (`lsode`, `dassl`, `daspk`) and in the `odepkg` package from Octave-Forge (`ode2r`, `ode5r`, `odebda`, `oders`, `odesx`). ■

8.11 What we haven't told you

For a complete derivation of the whole family of the Runge-Kutta methods we refer to [But87], [Lam91] and [QSS07, Chapter 11].

For derivation and analysis of multistep methods we refer to [Arn73] and [Lam91].

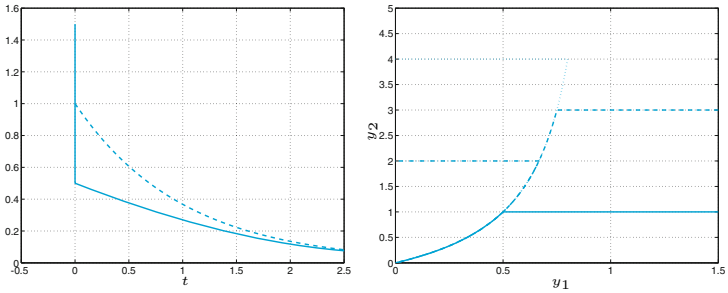

Figure 8.26. At left, numerical solutions ($y_1(t)$ *(continuous line)* and $y_2(t)$ *(dashed line)*) of system (8.84) with initial conditions $y_{1,0} = 1.5$, $y_{2,0} = 1$. At right, trajectories of (8.84) for several initial data $\mathbf{y}_0 = (y_{1,0}, y_{2,0})^T$: $\mathbf{y}_0 = (1.5, 1)^T$ *(continuous line)*, $(1.5, 3)^T$ *(dashed line)*, $(0, 2)^T$ *(dotted-dashed line)*, $(0, 4)^T$ *(dotted line)*. $\varepsilon = 10^{-6}$ in all simulations

8.12 Exercises

Exercise 8.1 Apply the backward Euler and forward Euler methods for the solution of the Cauchy problem

$$y' = \sin(t) + y,\ t \in (0, 1], \text{ with } y(0) = 0, \tag{8.85}$$

and verify that both converge with order 1.

Exercise 8.2 Consider the Cauchy problem

$$y' = -te^{-y},\ t \in (0, 1], \text{ with } y(0) = 0. \tag{8.86}$$

Apply the forward Euler method with $h = 1/100$ and estimate the number of exact significant digits of the approximate solution at $t = 1$ (use the property that the value of the exact solution is included between -1 and 0).

Exercise 8.3 The backward Euler method applied to problem (8.86) requires at each step the solution of the nonlinear equation: $u_{n+1} = u_n - ht_{n+1}e^{-u_{n+1}} = \phi(u_{n+1})$. The solution u_{n+1} can be obtained by the following fixed-point iteration:
for $k = 0, 1, \ldots$, compute $u_{n+1}^{(k+1)} = \phi(u_{n+1}^{(k)})$, with $u_{n+1}^{(0)} = u_n$.
Find under which restriction on h these iterations converge.

Exercise 8.4 Repeat Exercise 8.1 for the Crank-Nicolson method.

Exercise 8.5 Verify that the Crank-Nicolson method can be derived from the following integral form of the Cauchy problem (8.5)

$$y(t) - y_0 = \int_{t_0}^{t} f(\tau, y(\tau))d\tau$$

provided that the integral is approximated by the trapezoidal formula (4.19).

Exercise 8.6 Solve the model problem (8.28) with $\lambda = -1+i$ by the forward Euler method and find the values of h for which we have absolute stability.

Exercise 8.7 Show that the Heun method defined in (8.64) is consistent. Write a MATLAB program to implement it for the solution of the Cauchy problem (8.85) and verify experimentally that the method has order of convergence equal to 2 with respect to h.

Exercise 8.8 Prove that the Heun method (8.64) is absolutely stable if $-2 < h\lambda < 0$ where λ is real and negative.

Exercise 8.9 Prove formula (8.34).

Exercise 8.10 Prove the inequality (8.39).

Exercise 8.11 Prove the inequality (8.40).

Exercise 8.12 Verify the consistency of the RK3 method (8.58). Write a MATLAB program to implement it for the solution of the Cauchy problem (8.85) and verify experimentally that the method has order of convergence equal to 3 with respect to h. The methods (8.64) and (8.58) stand at the base of the MATLAB program `ode23` for the solution of ordinary differential equations.

Exercise 8.13 Prove that the method (8.58) is absolutely stable if $-2.5 < h\lambda < 0$ where λ is real and negative.

Exercise 8.14 The *modified Euler method* is defined as follows:

$$u^*_{n+1} = u_n + hf(t_n, u_n),\ u_{n+1} = u_n + hf(t_{n+1}, u^*_{n+1}). \tag{8.87}$$

Find under which condition on h this method is absolutely stable.

Exercise 8.15 (Thermodynamics) Solve equation (8.1) by the Crank-Nicolson method and the Heun method when the body in question is a cube with side equal to 1 m and mass equal to 1 Kg. Assume that $T_0 = 180K$, $T_e = 200K$, $\gamma = 0.5$ and $C = 100J/(Kg/K)$. Compare the results obtained by using $h = 20$ and $h = 10$, for t ranging from 0 to 200 seconds.

Exercise 8.16 Use MATLAB to compute the region of absolute stability of the Heun method.

Exercise 8.17 Solve the Cauchy problem (8.16) by the Heun method and verify its order.

Exercise 8.18 The displacement $x(t)$ of a vibrating system represented by a body of a given weight and a spring, subjected to a resistive force proportional to the velocity, is described by the second-order differential equation $x'' + 5x' + 6x = 0$. Solve it by the Heun method assuming that $x(0) = 1$ and $x'(0) = 0$, for $t \in [0, 5]$.

Exercise 8.19 The motion of a frictionless Foucault pendulum is described by the system of two equations

$$x'' - 2\omega \sin(\Psi) y' + k^2 x = 0, \quad y'' + 2\omega \cos(\Psi) x' + k^2 y = 0,$$

where Ψ is the latitude of the place where the pendulum is located, $\omega = 7.29 \cdot 10^{-5}$ sec^{-1} is the angular velocity of the Earth, $k = \sqrt{g/l}$ with $g = 9.8$ m/sec^2 and l is the length of the pendulum. Apply the forward Euler method to compute $x = x(t)$ and $y = y(t)$ for t ranging between 0 and 300 seconds and $\Psi = \pi/4$.

Exercise 8.20 (Baseball trajectory) Using `ode23`, solve Problem 8.3 by assuming that the initial velocity of the ball be $\mathbf{v}(0) = v_0(\cos(\phi), 0, \sin(\phi))^T$, with $v_0 = 38$ m/s, $\phi = 1$ degree and an angular velocity equal to $180 \cdot 1.047198$ radiants per second. If $\mathbf{x}(0) = \mathbf{0}$, after how many seconds (approximately) will the ball touch the ground (i.e., $z = 0$)?

Exercise 8.21 (Chemical kinetics) Given the real values $y_{1,0}$, $y_{2,0}$ e $y_{3,0}$, the following system

$$\begin{cases} \dfrac{dy_1}{dt} = \dfrac{1}{\varepsilon}(-5y_1 - y_1y_2 + 5y_2^2 + y_3) + y_2y_3 - y_1, & t > 0 \\ \dfrac{dy_2}{dt} = \dfrac{1}{\varepsilon}(10y_1 - y_1y_2 - 10y_2^2 + y_3) - y_2y_3 + y_1, & t > 0 \\ \dfrac{dy_3}{dt} = \dfrac{1}{\varepsilon}(y_1y_2 - y_3) - y_2y_3 + y_1, & t > 0 \\ y_1(0) = y_{1,0} \\ y_2(0) = y_{2,0}, y_3(0) = y_{3,0}, \end{cases} \tag{8.88}$$

simulates the evolution of the concentration of three species in a chemical reaction. By fixing the initial datum $\mathbf{y}_0 = (1, 0.5, 0)^T$ and setting $\varepsilon = 10^{-2}$, solve system (8.88) with $t \in [0, 10]$, calling `ode23` and `ode23s`, then comment on the stiffness of the system. Finally, plot the computed solution in the phase space, for different values of the initial datum $\mathbf{y}_0 = (y_{1,0}, y_{2,0}, y_{3,0})^T$ with $0 \le y_{i,0} \le 1$ and $i = 1, 3$.