8.4. NUMERICAL STABILITY, IMPLICIT METHODS

In Section 8.1, we discussed the stability property of the initial value problem (8.7), which states that a small perturbation in the initial value leads to a small change in the solution. With numerical methods for solving the initial value problem, it is desirable to have a similar stability property. This means that for any sufficiently small stepsize h, a small change in the initial value will lead to a small change in the numerical solution. Indeed, such a stability property is closely related to the convergence property. A well-known result is that under hypotheses similar to those of Theorem 8.1.3 on the initial value problem (8.7), a numerical method with truncation errors of order 2 or greater is convergent if and only if it is stable. For the Euler method, the truncation errors T_{n+1} are of order 2 (cf. (8.28)). Convergence of the method has been established in Theorem 8.3.1. Stability of the method can be shown by an argument similar to that leading to Theorem 8.3.1. For another example on the relations between convergence and stability, we refer to Problem 13 for a numerical method that is neither convergent nor stable.

The stability mentioned above states that a stable numerical method is well behaved, provided that the stepsize h is sufficiently small. In actual computations, however, the stepsize h cannot be too small since a very small stepsize decreases the efficiency of the numerical method. As is seen from the discussion of Section 5.4, the accuracy of difference approximations, such as $[Y(x+h) - Y(x)]/h$ to the derivative $Y'(x)$, deteriorates when h is too small. Hence, for actual computations, what matters is the performance of the numerical method when h is not assumed to be *very small*. We need to further analyze the stability of numerical methods when h is not assumed to be small.

Examining the stability question for the general problem

$$Y'(x) = f(x, Y(x)), \qquad Y(x_0) = Y_0 \tag{8.44}$$

is too complicated. Instead, we examine the stability of numerical methods for the *model equation*

$$Y'(x) = \lambda Y(x) + g(x), \qquad Y(0) = Y_0 \tag{8.45}$$

whose exact solution can be found from (8.4). Questions regarding stability and convergence are more easily answered for this equation; and the answers to these questions can be shown to usually be the answers to those same questions for the more general equation (8.44).

Let $Y(x)$ be the solution of (8.45), and let $Y_\epsilon(x)$ be the solution with the perturbed initial data $Y_0 + \epsilon$

$$Y'_\epsilon(x) = \lambda Y_\epsilon(x) + g(x), \qquad Y_\epsilon(0) = Y_0 + \epsilon$$

Let $Z_\epsilon(x)$ denote the change in the solution

$$Z_\epsilon(x) = Y_\epsilon(x) - Y(x)$$

Then, subtracting (8.45) from the equation for $Y_\epsilon(x)$, we get

$$Z'_\epsilon(x) = \lambda Z_\epsilon(x), \qquad Z_\epsilon(0) = \epsilon$$

The solution is

$$Z_\epsilon(x) = \epsilon e^{\lambda x}$$

Typically in applications, we are interested in the case that $\lambda < 0$ or that λ is complex with a negative real part. In such a case, $Z_\epsilon(x)$ will go to zero as $n \to \infty$ and, thus, the effect of the ϵ perturbation dies out for large values of x. We would like the same behavior to hold for the numerical method that is being applied to (8.45).

By considering the function $Z_\epsilon(x)/\epsilon$ instead of $Z_\epsilon(x)$, we obtain the following model problem that is commonly used to test the performance of various numerical methods:

$$\begin{cases} Y' = \lambda Y, & x > 0 \\ Y(0) = 1 \end{cases} \tag{8.46}$$

In the following, when we refer to the model problem (8.46), we always assume the constant $\lambda < 0$ or λ has a negative real part. The true solution of the problem (8.46) is

$$Y(x) = e^{\lambda x} \tag{8.47}$$

which decays exponentially in x since the parameter λ has a negative real part.

The kind of stability property we would like for a numerical method is that when it is applied to (8.46), the numerical solution satisfies

$$y_h(x_n) \to 0 \qquad \text{as } x_n \to \infty \tag{8.48}$$

for any choice of the stepsize h. This property is called *absolute stability*. The set of values $h\lambda$, considered as a subset of the complex plane, for which $y_n \to 0$ as $h \to 0$, is called the *region of absolute stability* of the numerical method.

Let us examine the performance of the Euler method on the model problem (8.46). We have

$$y_{n+1} = y_n + h\,\lambda\, y_n = (1 + h\,\lambda)\, y_n, \qquad n \geq 0, \quad y_0 = 1$$

By an inductive argument, it is not difficult to find

$$y_n = (1 + h\,\lambda)^n, \qquad n \geq 0 \tag{8.49}$$

Note that for a fixed grid point $x_n = n\,h \equiv \overline{x}$, as $n \to \infty$,

$$y_n = \left(1 + \frac{\lambda\,\overline{x}}{n}\right)^n \to e^{\lambda \overline{x}}$$

The limiting behavior is obtained using l'Hospital's rule from calculus. This confirms the convergence of the Euler method. We emphasize that this is an asymptotic property in the sense that it is valid in the limit $h \to 0$.

From the formula (8.49), we see that $y_n \to 0$ as $n \to \infty$ if and only if

$$|1 + h\lambda| < 1$$

For λ real and negative, the condition becomes

$$h\lambda > -2 \tag{8.50}$$

This sets a restriction on the range of h we can take to apply Euler's method: $0 < h < (-2)/\lambda$.

Example 8.4.1 Consider the model problem with $\lambda = -100$. Then the Euler method will perform well only when $h < 2 \cdot 100^{-1} = 0.02$. The true solution $Y(x) = e^{-100x}$ at $x = 0.2$ is 2.061×10^{-9}. Table 8.4 lists the Euler solution at $x = 0.2$ for several values of h. ■

8.4.1 The Backward Euler Method

Now we consider a numerical method that has the property (8.48) for any stepsize h when applied to the model problem (8.46). Such a method is said to be *absolutely stable*. In the derivation of the Euler method, we used the forward difference approximation

$$Y'(x) \approx \frac{1}{h}[Y(x+h) - Y(x)]$$

Let us use, instead, the backward difference approximation

$$Y'(x) \approx \frac{1}{h}[Y(x) - Y(x-h)]$$

Table 8.4. Euler's Solution at $x = 0.2$ for Several Values of h

h	$y_h(0.2)$
0.1	81
0.05	256
0.02	1
0.01	0
0.001	7.06E − 10

Then the differential equation $Y'(x) = f(x, Y(x))$ at $x = x_n$ is discretized as

$$y_n = y_{n-1} + h\, f(x_n, y_n)$$

Shifting the index by 1, we then obtain the backward Euler method

$$\begin{cases} y_{n+1} = y_n + h\, f(x_{n+1}, y_{n+1}), & 0 \le n \le N-1 \\ y_0 = Y_0 \end{cases} \tag{8.51}$$

Like the Euler method, the backward Euler method is of first-order accuracy, and a convergence result similar to Theorem 8.3.1 holds. Also, an asymptotic error expansion of the form (8.38) is valid.

Let us show that the backward Euler method has the desired property (8.48) on the model problem (8.46). We have

$$\begin{aligned} y_{n+1} &= y_n + h\lambda\, y_{n+1} \\ y_{n+1} &= (1 - h\lambda)^{-1} y_n, \qquad n \ge 0, \quad y_0 = 1 \end{aligned}$$

Then

$$y_n = (1 - h\lambda)^{-n} \tag{8.52}$$

For any stepsize $h > 0$, $|1 - h\lambda| > 1$ and so $y_n \to 0$ as $n \to \infty$. Continuing Example 8.4.1, we have the following table for numerical results from the backward Euler method.

A comparison between Table 8.4 and Table 8.5 reveals that the backward Euler method is substantially better than the Euler method on the model problem (8.46).

The major difference between the two methods is that for the backward Euler method, at each time step, we need to solve a nonlinear algebraic equation

$$y_{n+1} = y_n + h\, f(x_{n+1}, y_{n+1}) \tag{8.53}$$

for y_{n+1}. Methods in which y_{n+1} must be found by solving a rootfinding problem are called *implicit methods*, since y_{n+1} is defined implicitly. In contrast, methods that give y_{n+1} directly are called *explicit methods*. Euler's method is an explicit method, whereas

Table 8.5. Backward Euler's Solution at $x = 0.2$ for Several Values of h

h	$y_h(0.2)$
0.1	8.26E − 3
0.05	7.72E − 4
0.02	1.69E − 5
0.01	9.54E − 7
0.001	5.27E − 9

the backward Euler method is an implicit method. Under the Lipschitz continuity assumption of the function $f(x, y)$ with respect to y, it can be shown by applying Theorem 3.4.2 that if h is small enough, the equation (8.53) has a unique solution.

The rootfinding methods of Chapter 3 can be applied to (8.53) to find its root y_{n+1}; but usually that will be a very time-consuming process. Instead, (8.53) is usually solved by a simple iteration technique. Given an initial guess $y_{n+1}^{(0)} \approx y_{n+1}$, define $y_{n+1}^{(1)}$, $y_{n+1}^{(2)}$, etc., by

$$y_{n+1}^{(j+1)} = y_n + h\, f(x_{n+1}, y_{n+1}^{(j)}), \qquad j = 0, 1, 2, \ldots \tag{8.54}$$

It can be shown that if h is sufficiently small, then the iterates $y_{n+1}^{(j)}$ will converge to y_{n+1} as $j \to \infty$. Subtracting (8.54) from (8.53) gives us

$$\begin{aligned} y_{n+1} - y_{n+1}^{(j+1)} &= h\,[f(x_{n+1}, y_{n+1}) - f(x_{n+1}, y_{n+1}^{(j)})] \\ y_{n+1} - y_{n+1}^{(j+1)} &\approx h \cdot \frac{\partial f(x_{n+1}, y_{n+1})}{\partial z}[y_{n+1} - y_{n+1}^{(j)}] \end{aligned}$$

The last formula is obtained by applying the mean value theorem to $f(x_{n+1}, z)$, considered a function of z. This formula gives a relation between the error in successive iterates. Therefore, if

$$\left| h \cdot \frac{\partial f(x_{n+1}, y_{n+1})}{\partial z} \right| < 1 \tag{8.55}$$

then the errors will converge to zero, as long as the initial guess $y_{n+1}^{(0)}$ is good enough.

The above iteration method (8.54) and its analysis are simply a special case of the theory of fixed point iteration from Section 3.4. In the notation of that earlier material, $\alpha = y_{n+1}$ is the fixed point, and

$$g(z) \equiv y_n + h\, f(x_{n+1}, z)$$

The convergence condition (8.55) is simply the earlier condition $|g'(\alpha)| < 1$ of (3.47). The remaining results of Section 3.4 can be applied to (8.53) and (8.54), but there is little benefit to doing so here.

In practice, one uses a good initial guess $y_{n+1}^{(0)}$, and one chooses an h that is so small that the quantity in (8.55) is much less than 1. Then the error $y_{n+1} - y_{n+1}^{(j)}$ decreases rapidly to a small quantity, and usually only one iterate needs to be computed. The usual choice of the initial guess $y_{n+1}^{(0)}$ for (8.54) is based on the Euler method

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n)$$

It is called a *predictor formula*, as it predicts the root of the implicit method.

For many equations, it is usually sufficient to do the iteration (8.54) once. Thus, a practical way to implement the backward Euler method is to do the following one-point

iteration for solving (8.53) approximately:

$$\bar{y}_{n+1} = y_n + h\, f(x_{n+1}, y_n)$$
$$y_{n+1} = y_n + h\, f(x_{n+1}, \bar{y}_{n+1})$$

The resulting numerical method is then given by the formula

$$y_{n+1} = y_n + h\, f(x_{n+1}, y_n + h\, f(x_{n+1}, y_n)) \tag{8.56}$$

It can be shown that this method is still of first-order accuracy. However, it is no longer absolutely stable (cf. Problem 1).

MATLAB PROGRAM. Now we turn to an implementation of the backward Euler method. At each step, with y_n available from the previous step, we use the Euler method to compute an estimate of y_{n+1}:

$$y_{n+1}^{(1)} = y_n + hf(x_n, y_n)$$

Then we use the trapezoidal formula to do the iteration

$$y_{n+1}^{(k+1)} = y_n + h\, f(x_{n+1}, y_{n+1}^{(k)})$$

until the difference between successive values of the iterates is sufficiently small, indicating a sufficiently accurate approximation of the solution y_{n+1}. To prevent an infinite loop of iteration, we require the iteration to stop if 10 iteration steps are taken without reaching a satisfactory solution; and in this latter case, an error message will be displayed.

```
function [x,y] = euler_back(x0,y0,x_end,h,fcn,tol)
%
% function [x,y] = euler_back(x0,y0,x_end,h,fcn,tol)
%
% Solve the initial value problem
%    y' = f(x,y),  x0 <= x <= b,  y(x0)=y0
% Use the backward Euler method with a stepsize of h.
% The user must supply an m-file to define the
% derivative f, with some name, say, 'deriv.m', and a
% first line of the form
%     function ans=deriv(x,y)
% tol is the user supplied bound on the difference
% between successive values of the trapezoidal
% iteration.  A sample call would be
%   [t,z]=euler_back(t0,z0,b,delta,'deriv',1.0e-3)
%
% Output:
```

```
% The routine euler_back will return two vectors,
% x and y. The vector x will contain the node points
%   x(1)=x0, x(j)=x0+(j-1)*h, j=1,2,...,N
% with
%   x(N) <= x_end,  x(N)+h > x_end
% The vector y will contain the estimates of the
% solution Y at the node points in x.
%

% Initialize.
n = fix((x_end-x0)/h)+1;
x = linspace(x0,x_end,n)';
y = zeros(n,1);
y(1) = y0;
i = 2;
% advancing
while i <= n
%
% forward Euler estimate
%
  yt1 = y(i-1)+h*feval(fcn,x(i-1),y(i-1));
% one-point iteration
  count = 0;
  diff = 1;
  while diff > tol & count < 10
    yt2 = y(i-1) + h*feval(fcn,x(i),yt1);
    diff = abs(yt2-yt1);
    yt1 = yt2;
    count = count +1;
  end
  if count >= 10
    disp('Not converging after 10 steps at x = ')
    fprintf('%5.2f\n', x(i))
  end
  y(i) = yt2;
  i = i+1;
end
```

8.4.2 The Trapezoidal Method

One main drawback of both the Euler method and the backward Euler method is the low convergence order. Next we present a method that has a higher convergence order, and at the same time, the stability property (8.48) is valid for any stepsize h in solving the model problem (8.46).

We integrate the differential equation

$$Y'(x) = f(x, Y(x))$$

from x_n to x_{n+1}:

$$Y(x_{n+1}) = Y(x_n) + \int_{x_n}^{x_{n+1}} f(x, Y(x))\, dx$$

and use the trapezoidal rule to approximate the integral

$$Y(x_{n+1}) \approx Y(x_n) + \frac{h}{2}[f(x_n, Y(x_n)) + f(x_{n+1}, Y(x_{n+1}))]$$

By equating both sides, we then obtain the trapezoidal method for solving the initial value problem (8.7):

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})], & n \geq 0 \\ y_0 = Y_0 \end{cases} \tag{8.57}$$

It can be shown that the method is of second-order accuracy, and it is absolutely stable.

Notice that the trapezoidal method is an implicit method. In a general step, y_{n+1} is found from the equation

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})] \tag{8.58}$$

and it can be solved directly in only a small percentage of cases. The discussion for the solution of the backward Euler equation (8.53) applies to the solution of the equation (8.58), with a slight variation. The iteration formula (8.54) is now replaced by

$$y_{n+1}^{(j+1)} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(j)})], \qquad j = 0, 1, 2, \ldots \tag{8.59}$$

And if h is sufficiently small, then the iterates $y_{n+1}^{(j)}$ will converge to y_{n+1} as $j \to \infty$. The convergence condition (8.55) is replaced by

$$\left| \frac{h}{2} \cdot \frac{\partial f(x_{n+1}, y_{n+1})}{\partial z} \right| < 1 \tag{8.60}$$

Note that the condition (8.60) is easier to satisfy than (8.55), indicating the trapezoidal method is easier to use than the backward Euler method.

The usual choice of the initial guess $y_{n+1}^{(0)}$ for (8.59) is based on the Euler method

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n) \tag{8.61}$$

or an Adams–Bashforth method of order 2 (cf. Section 8.6)

$$y_{n+1}^{(0)} = y_n + \frac{h}{2}[3f(x_n, y_n) - f(x_{n-1}, y_{n-1})] \tag{8.62}$$

These are called *predictor formulas*. In either of the two above cases, compute $y_{n+1}^{(1)}$ from (8.59) and accept it as the root y_{n+1}. With both methods of choosing $y_{n+1}^{(0)}$, it can be shown that the global error in the resulting solution $\{y_h(x_n)\}$ is still $O(h^2)$. If the Euler predictor (8.61) is used to define $y_{n+1}^{(0)}$, and we accept $y_{n+1}^{(1)}$ as the value of y_{n+1}, then the resulting new scheme is

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_n + h\,f(x_n, y_n))] \tag{8.63}$$

known as *Heun's method*. The Heun method is still of second-order accuracy. However, it is no longer absolutely stable.

MATLAB PROGRAM. In our implementation of the trapezoidal method, at each step, with y_n available from the previous step, we use the Euler method to compute an estimate of y_{n+1}:

$$y_{n+1}^{(1)} = y_n + hf(x_n, y_n)$$

Then we use the trapezoidal formula to do the iteration

$$y_{n+1}^{(k+1)} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(k)})\right]$$

until the difference between successive values of the iterates is sufficiently small, indicating a sufficiently accurate approximation of the solution y_{n+1}. To prevent an infinite loop of iteration, we require the iteration to stop if 10 iteration steps are taken without reaching a satisfactory solution; and in this latter case, an error message will be displayed.

```
function [x,y] = trapezoidal(x0,y0,x_end,h,fcn,tol)
%
% function [x,y] = trapezoidal(x0,y0,x_end,h,fcn,tol)
%
% Solve the initial value problem
%    y' = f(x,y),  x0 <= x <= b,  y(x0)=y0
% Use trapezoidal method with a stepsize of h.  The
% user must supply an m-file to define the derivative
% f, with some name, say, 'deriv.m', and a first line
% of the form
%   function ans=deriv(x,y)
```

```
% tol is the user supplied bound on the difference
% between successive values of the trapezoidal
% iteration. A sample call would be
%   [t,z]=trapezoidal(t0,z0,b,delta,'deriv',1e-3)
%
% Output:
% The routine trapezoidal will return two vectors,
% x and y. The vector x will contain the node points
%   x(1) = x0, x(j) = x0+(j-1)*h, j=1,2,...,N
% with
%   x(N) <= x_end,  x(N)+h > x_end
% The vector y will contain the estimates of the
% solution Y at the node points in x.
%

% Initialize.
n = fix((x_end-x0)/h)+1;
x = linspace(x0,x_end,n)';
y = zeros(n,1);
y(1) = y0;
i = 2;
% advancing
while i <= n
  fyt = feval(fcn,x(i-1),y(i-1));
%
% Euler estimate
%
  yt1 = y(i-1)+h*fyt;
% trapezoidal iteration
  count = 0;
  diff = 1;
  while diff > tol & count < 10
    yt2 = y(i-1) + h*(fyt+feval(fcn,x(i),yt1))/2;
    diff = abs(yt2-yt1);
    yt1 = yt2;
    count = count +1;
  end
  if count >= 10
    disp('Not converging after 10 steps at x = ')
    fprintf('%5.2f\n', x(i))
  end
  y(i) = yt2;
  i = i+1;
end
```

Table 8.6. Euler's Method for (8.64)

λ	x	Error: $h = 0.5$	Error: $h = 0.1$	Error: $h = 0.01$
-1	1	$-2.46\text{E}-1$	$-4.32\text{E}-2$	$-4.22\text{E}-3$
	2	$-2.55\text{E}-1$	$-4.64\text{E}-2$	$-4.55\text{E}-3$
	3	$-2.66\text{E}-2$	$-6.78\text{E}-3$	$-7.22\text{E}-4$
	4	$2.27\text{E}-1$	$3.91\text{E}-2$	$3.78\text{E}-3$
	5	$2.72\text{E}-1$	$4.91\text{E}-2$	$4.81\text{E}-3$
-10	1	$3.98\text{E}-1$	$-6.99\text{E}-3$	$-6.99\text{E}-4$
	2	$6.90\text{E}+0$	$-2.90\text{E}-3$	$-3.08\text{E}-4$
	3	$1.11\text{E}+2$	$3.86\text{E}-3$	$3.64\text{E}-4$
	4	$1.77\text{E}+3$	$7.07\text{E}-3$	$7.04\text{E}-4$
	5	$2.83\text{E}+4$	$3.78\text{E}-3$	$3.97\text{E}-4$
-50	1	$3.26\text{E}+0$	$1.06\text{E}+3$	$-1.39\text{E}-4$
	2	$1.88\text{E}+3$	$1.11\text{E}+9$	$-5.16\text{E}-5$
	3	$1.08\text{E}+6$	$1.17\text{E}+15$	$8.25\text{E}-5$
	4	$6.24\text{E}+8$	$1.23\text{E}+21$	$1.41\text{E}-4$
	5	$3.59\text{E}+11$	$1.28\text{E}+27$	$7.00\text{E}-5$

Example 8.4.2 Consider the problem

$$Y'(x) = \lambda Y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x), \qquad Y(0) = 1 \tag{8.64}$$

whose true solution is $Y(x) = \sin(x) + \cos(x)$. Euler's method is used for the numerical solution, and the results for several values of λ and h are given in Table 8.6. Note that according to the formula (8.28) for the truncation error,

$$T_{n+1} = \frac{h^2}{2} Y''(\xi_n)$$

The solution will not depend on λ, since $Y(x)$ does not depend on λ. But the actual global error depends strongly on λ, as illustrated in the table; and the behavior of the global error is directly linked to the size of λh and, thus, to the size of the stability region for Euler's method. The error is small, provided that λh is sufficiently small. The cases of an unstable and rapid growth in the error are exactly the cases in which λh is outside the range (8.50).

We then apply the backward Euler method and the trapezoidal method to the solution of the problem (8.64). The results are shown in Tables 8.7 and 8.8, with the stepsize $h = 0.5$. The error varies with λ, but there are no stability problems, in contrast to the Euler method. The solutions of the backward Euler method and the trapezoidal method for y_{n+1} were done exactly. This is possible because the differential equation is linear in Y. The fixed point iterations (8.54) and (8.59) do not converge when $|\lambda h|$ is large. ∎

Table 8.7. Backward Euler Solution for (8.64); $h = 0.5$

x	Error: $\lambda = -1$	Error: $\lambda = -10$	Error: $\lambda = -50$
2	2.08E − 1	1.97E − 2	3.60E − 3
4	−1.63E − 1	−3.35E − 2	−6.94E − 3
6	−7.04E − 2	8.19E − 3	2.18E − 3
8	2.22E − 1	2.67E − 2	5.13E − 3
10	−1.14E − 1	−3.04E − 2	−6.45E − 3

Table 8.8. Trapezoidal Solution for (8.64); $h = 0.5$

x	Error: $\lambda = -1$	Error: $\lambda = -10$	Error: $\lambda = -50$
2	−1.13E − 2	−2.78E − 3	−7.91E − 4
4	−1.43E − 2	−8.91E − 5	−8.91E − 5
6	2.02E − 2	2.77E − 3	4.72E − 4
8	−2.86E − 3	−2.22E − 3	−5.11E − 4
10	−1.79E − 2	−9.23E − 4	−1.56E − 4

Equations of the form (8.45)

$$Y'(x) = \lambda Y(x) + g(x)$$

with λ negative but large in magnitude are so-called *stiff differential equations*. A stiff differential equation is characterized by the property that when it is solved by a numerical scheme without absolute stability, the stepsize h is usually forced to be excessively small in order for the scheme to generate reasonable approximate solution. See the discussions in Examples 8.4.1 and 8.4.2 for the restriction of the size of h in using the Euler method to solve the model equations (8.45) and (8.64). For stiff differential equations, one must use a numerical method that is absolutely stable or has a large region of absolute stability. Usually, implicit methods are preferred to the explicit ones in solving stiff differential equations. The MATLAB built-in function `ode23tb` is especially designed to solve stiff differential equations. Its use is similar to that of another MATLAB built-in function `ode45`, illustrated in Subsection 8.5.3.

PROBLEMS

1. Show that the method defined by the formula (8.56) is not absolutely stable.
2. Show that the trapezoidal method (8.57) is absolutely stable, but the scheme (8.63) is not.
3. Use the backward Euler's method to solve Problem 2 of Section 8.2.
4. Use the trapezoidal method to solve Problem 2 of Section 8.2.
5. Apply the backward Euler method to solve the initial value problem in Problem 6 of Section 8.3 for $\alpha = 2.5, 1.5, 1.1$, with $h = 0.2, 0.1, 0.05$. Compute the error

10. Consider solving the pendulum equation (8.112) with $l = 1$ and $g = 32.2$ ft/sec^2. For the initial values, choose $0 < \theta(0) \leq \pi/2$, $\theta'(0) = 0$. Use Euler's method to solve (8.113), and experiment with various values of h so as to obtain a suitably small error in the computed solution. Graph t versus $\theta(t)$, t versus $\theta'(t)$, and $\theta(t)$ versus $\theta'(t)$. Does the motion appear to be periodic in time?

8.8. FINITE DIFFERENCE METHOD FOR TWO-POINT BOUNDARY VALUE PROBLEMS

In the previous section, we have seen that the initial value problem of the second-order equation

$$Y'' = f(x, Y, Y') \tag{8.124}$$

can be reformulated as an initial value problem of a system of first-order equations. In this section, we consider the solution of another type of problems for the second-order equation (8.124), where conditions on the solution Y are given at two distinct x values. Such problems are called *two-point boundary value problems*. For simplicity, our discussion in this section will be focused on the following boundary value problem of a second-order *linear* equation:

$$\begin{cases} Y''(x) = p(x)\,Y'(x) + q(x)\,Y(x) + r(x), & a \leq x \leq b \\ Y(a) = g_1, \quad Y(b) = g_2 \end{cases} \tag{8.125}$$

The conditions $Y(a) = g_1$ and $Y(b) = g_2$ are called the *boundary conditions*. Boundary conditions involving the derivative of the unknown function are also common in applications, and we will comment on the finite difference approximations of such boundary conditions later in the section.

We assume the given functions p, q, and r are continuous on $[a, b]$. A standard theoretical result states that if $q(x) > 0$ for $x \in [a, b]$, then the boundary value problem (8.125) has a unique solution. We will assume the problem has a unique smooth solution Y.

The main feature of the finite difference method is to obtain discrete equations by replacing derivatives with appropriate finite divided differences. We derive a finite difference system for the boundary value problem (8.125) in three steps.

In the first step, we discretize the domain of the problem: the interval $[a, b]$. Let N be a positive integer, and divide the interval $[a, b]$ into N equal parts:

$$[a, b] = [x_0, x_1] \cup [x_1, x_2] \cup \cdots \cup [x_{N-1}, x_N]$$

where $a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b$ are the grid (or node) points. Denote $h = (b - a)/N$, called the *stepsize*. Then the node points are given by

$$x_i = a + i\,h, \qquad 0 \leq i \leq N$$

Nonuniform partition of the interval is also possible, and is in fact preferred if the solution of the boundary value problem (8.125) changes much more rapidly in some part of the interval than the remaining part. We restrict ourselves to the case of uniform partitions for simplicity of exposition. We use the notation $p_i = p(x_i)$, $q_i = q(x_i)$, $r_i = r(x_i)$, $0 \le i \le N$, and denote y_i, $0 \le i \le N$, to be numerical approximations of the true solution values $Y_i = Y(x_i)$, $0 \le i \le N$.

In the second step, we discretize the differential equation at the interior node points $x_1, \ldots, x_{N-1}$. For this purpose, let us recall the following difference approximation formulas (cf. (5.85), (5.92)):

$$Y'(x_i) = \frac{Y_{i+1} - Y_{i-1}}{2h} + O(h^2) \tag{8.126}$$

$$Y''(x_i) = \frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} + O(h^2) \tag{8.127}$$

Then the differential equation at $x = x_i$ becomes

$$\frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} = p_i \frac{Y_{i+1} - Y_{i-1}}{2h} + q_i Y_i + r_i + O(h^2) \tag{8.128}$$

Dropping the remainder term $O(h^2)$ and replacing Y_i by y_i, we obtain the difference equations

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i + r_i, \qquad 1 \le i \le N-1$$

which can be rewritten as

$$-\left(1 + \frac{h}{2} p_i\right) y_{i-1} + (2 + h^2 q_i) y_i + \left(\frac{h}{2} p_i - 1\right) y_{i+1} = -h^2 r_i, \qquad 1 \le i \le N-1 \tag{8.129}$$

The third step is devoted to the treatment of the boundary conditions. The difference equations (8.129) consist of $(N-1)$ equations for $(N+1)$ unknowns $y_0, y_1, \ldots, y_N$. We need two more equations and they come from discretization of the boundary conditions. For the model problem (8.125), the discretization of the boundary conditions is straightforward:

$$y_0 = g_1, \qquad y_N = g_2 \tag{8.130}$$

The equations (8.129) and (8.130) together form a linear system. Since the values of y_0 and y_N are explicitly given in (8.130), we can eliminate y_0 and y_N from the linear system. With $y_0 = g_1$, we can rewrite the equation in (8.129) with $i = 1$ as

$$(2 + h^2 q_1)\, y_1 + \left(\frac{h}{2} p_1 - 1\right) y_2 = -h^2 r_1 + \left(1 + \frac{h}{2} p_1\right) g_1 \tag{8.131}$$

Similarly, from the equation in (8.128) with $i = N-1$, we obtain

$$-\left(1+\frac{h}{2}p_{N-1}\right)y_{N-2}+(2+h^2q_{N-1})\,y_{N-1}=-h^2r_{N-1}+\left(1-\frac{h}{2}p_{N-1}\right)g_2 \tag{8.132}$$

So finally, the finite difference system for the unknown numerical solution vector $\boldsymbol{y}=[y_1,\ldots,y_{N-1}]^T$ is

$$A\boldsymbol{y}=\boldsymbol{b} \tag{8.133}$$

where

$$A=\begin{bmatrix} 2+h^2q_1 & \frac{h}{2}p_1-1 & & & \\ -\left(1+\frac{h}{2}p_2\right) & 2+h^2q_2 & \frac{h}{2}p_2-1 & & \\ & \ddots & \ddots & \ddots & \\ & & -\left(1+\frac{h}{2}p_{N-2}\right) & 2+h^2q_{N-2} & \frac{h}{2}p_{N-2}-1 \\ & & & -\left(1+\frac{h}{2}p_{N-1}\right) & 2+h^2q_{N-1} \end{bmatrix} \tag{8.134}$$

is the coefficient matrix and

$$\boldsymbol{b}=\left[-h^2r_1+\left(1+\frac{h}{2}p_1\right)g_1,\,-h^2r_2,\ldots,-h^2r_{N-2},\,-h^2r_{N-1}+\left(1-\frac{h}{2}p_{N-1}\right)g_2\right]^T \tag{8.135}$$

is the right-hand-side vector.

It can be shown that if the true solution $Y(x)$ is sufficiently smooth, say, it has continuous derivatives up to order 4, then the difference scheme (8.133) is a second-order method, that is,

$$\max_{0\le i\le N}|Y(x_i)-y_i|=O(h^2)$$

Moreover, the following asymptotic error expansion holds:

$$Y(x_i)-y_h(x_i)=h^2D(x_i)+O(h^4),\qquad 0\le i\le N \tag{8.136}$$

for some function $D(x)$ independent of h. The Richardson extrapolation formula for this case is

$$\tilde{y}_h(x_i)=\frac{4\,y_h(x_i)-y_{2h}(x_i)}{3} \tag{8.137}$$

and we have

$$Y(x_i)-\tilde{y}_h(x_i)=O(h^4) \tag{8.138}$$

The linear system (8.133) is tridiagonal. So it is natural to use the algorithm developed in Section 6.4 to solve the system (8.133).

MATLAB PROGRAM. The following MATLAB code implements the difference method (8.133) for solving the problem (8.125):

```
function z = ODEBVP(p,q,r,a,b,ga,gb,N)
% A program to solve the two-point boundary
% value problem
%   y"=p(x)y'+q(x)y+r(x),  a<x<b
%   y(a)=ga,  y(b)=gb
% Input
%   p, q, r: coefficient functions
%   a, b: the end-points of the interval
%   ga, gb: the prescribed function values
%           at the end-points
%   N: number of subintervals
% Output
%   z = [ xx yy ]: xx is an (N+1) column vector
%                  of the node points
%                yy is an (N+1) column vector of
%                  the solution values
% A sample call would be
%   z=ODEBVP('p','q','r',a,b,ga,gb,100)
% The user must provide m-files to define the
% functions p, q and r.
%
% Other MATLAB program called: tridiag.m
%
% Initialization
N1 = N+1;
h = (b-a)/N;
h2 = h*h;
xx = linspace(a,b,N1)';
yy = zeros(N1,1);
yy(1) = ga;
yy(N1) = gb;
% Define the subdiagonal avec, main diagonal bvec,
% superdiagonal cvec
pp(2:N) = feval(p,xx(2:N));
avec(2:N-1) = -1-(h/2)*pp(3:N);
bvec(1:N-1) = 2+h2*feval(q,xx(2:N));
cvec(1:N-2) = -1+(h/2)*pp(2:N-1);
% Define the right-hand-side vector fvec
fvec(1:N-1) = -h2*feval(r,xx(2:N));
```

```
fvec(1) = fvec(1)+(1+h*pp(2)/2)*ga;
fvec(N-1) = fvec(N-1)+(1-h*pp(N)/2)*gb;
% Solve the tridiagonal system
yy(2:N) = tridiag(avec,bvec,cvec,fvec,N-1,0);
z = [xx'; yy']';
```

Example 8.8.1 Consider the boundary value problem

$$\begin{cases} Y'' = -\dfrac{2x}{1+x^2}Y' + Y + \dfrac{2}{1+x^2} - \log(1+x^2), \quad 0 \le x \le 1 \\ Y(0) = 0, \quad Y(1) = \log(2) \end{cases} \tag{8.139}$$

The true solution is $Y(x) = \log(1+x^2)$.

In Table 8.24, we report the finite difference solution errors $Y - y_h$ at selected node points for several values of h. In Table 8.25, we report the errors of the extrapolated solutions $Y - (4y_h - y_{2h})/3$ at the same node points and the associated ratios of the errors for different stepsizes. The column marked Ratio next to the column of the solution errors for a stepsize h consists of the ratios of the solution errors for the stepsize h with those for the stepsize $2h$. We clearly observe an error reduction of a factor of around 4 when the stepsize is halved, indicating a second-order convergence of the method. There is a dramatic improvement in the solution accuracy through extrapolation. The extrapolated solution $\tilde{y}_h$ with $h = 1/40$ is much more accurate than the solution y_h with $h = 1/160$. Note that the cost of obtaining $\tilde{y}_h$ with $h = 1/40$ is substantially smaller than that for y_h with $h = 1/160$.

Also observe that for the extrapolated solution $\tilde{y}_h$, the error decreases by approximately a factor of 16 when h is halved. Indeed, it can be shown that if the true solution $Y(x)$ is six times continuously differentiable, then we can improve the asymptotic error

Table 8.24. Numerical Errors $Y(x) - y_h(x)$ for Solving the Problem (8.139)

x	$h = 1/20$	$h = 1/40$	Ratio	$h = 1/80$	Ratio	$h = 1/160$	Ratio
0.1	5.10E − 5	1.27E − 5	4.00	3.18E − 6	4.00	7.96E − 7	4.00
0.2	7.84E − 5	1.96E − 5	4.00	4.90E − 6	4.00	1.22E − 6	4.00
0.3	8.64E − 5	2.16E − 5	4.00	5.40E − 6	4.00	1.35E − 6	4.00
0.4	8.08E − 5	2.02E − 5	4.00	5.05E − 6	4.00	1.26E − 6	4.00
0.5	6.73E − 5	1.68E − 5	4.00	4.21E − 6	4.00	1.05E − 6	4.00
0.6	5.08E − 5	1.27E − 5	4.00	3.17E − 6	4.00	7.94E − 7	4.00
0.7	3.44E − 5	8.60E − 6	4.00	2.15E − 6	4.00	5.38E − 7	4.00
0.8	2.00E − 5	5.01E − 6	4.00	1.25E − 6	4.00	3.13E − 7	4.00
0.9	8.50E − 6	2.13E − 6	4.00	5.32E − 7	4.00	1.33E − 7	4.00

Table 8.25. Extrapolation Errors for Solving the Problem (8.139)

x	$h = 1/40$	$h = 1/80$	Ratio	$h = 1/160$	Ratio
0.1	−9.23E − 09	−5.76E − 10	16.01	−3.60E − 11	16.00
0.2	−1.04E − 08	−6.53E − 10	15.99	−4.08E − 11	15.99
0.3	−6.60E − 09	−4.14E − 10	15.96	−2.59E − 11	15.98
0.4	−1.18E − 09	−7.57E − 11	15.64	−4.78E − 12	15.85
0.5	3.31E − 09	2.05E − 10	16.14	1.28E − 11	16.06
0.6	5.76E − 09	3.59E − 10	16.07	2.24E − 11	16.04
0.7	6.12E − 09	3.81E − 10	16.04	2.38E − 11	16.03
0.8	4.88E − 09	3.04E − 10	16.03	1.90E − 11	16.03
0.9	2.67E − 09	1.67E − 10	16.02	1.04E − 11	16.03

expansion (8.136) to

$$Y(x_i) - y_h(x_i) = h^2 D_1(x_i) + h^4 D_2(x_i) + O(h^6)$$

Then (8.137) is replaced by

$$Y(x_i) - \tilde{y}_h(x_i) = -4\,h^4 D_2(x_i) + O(h^6)$$

Therefore, we can perform an extrapolation procedure also on $\tilde{y}_h$ to get an even more accurate numerical solution through the formula

$$Y(x_i) - \frac{16\,\tilde{y}_h(x_i) - \tilde{y}_{2h}(x_i)}{15} = O(h^6)$$

As an example, at $x_i = 0.5$, with $h = 1/80$, the further extrapolated solution has an error approximately equal to −1.88E−12. ■

So far, our discussions have focused on the solution of the particular boundary value problem (8.125). Let us make some remarks on the various extensions.

In principle, difference schemes for solving the more general equation (8.124) can be derived similarly. For example, if we use the formulas (8.126) and (8.127), the equation (8.124) at an interior node point x_i can be approximated by the difference equation

$$\frac{y_{i+1} - 2\,y_i + y_{i-1}}{h^2} = f\left(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2\,h}\right)$$

The treatment of boundary conditions involving the derivative of the unknown is a little bit more involved. Assume the boundary condition at $x = b$ is replaced by

$$Y'(b) + k\,Y(b) = g_2 \tag{8.140}$$

One obvious possibility is to approximate $Y'(b)$ by $(Y_N - Y_{N-1})/h$. However,

$$Y'(b) - \frac{Y_N - Y_{N-1}}{h} = O(h)$$

and the accuracy of this approximation is one order lower than the remainder term $O(h^2)$ in (8.128). As a result, the corresponding difference solution with the following discrete boundary condition:

$$\frac{y_N - y_{N-1}}{h} + k\, y_N = g_2 \tag{8.141}$$

will have an accuracy of $O(h)$ only. To retain the second-order convergence of the difference solution, we need to approximate the boundary condition (8.140) more accurately. One such treatment is based on the formula

$$Y'(b) = \frac{3\,Y_N - 4\,Y_{N-1} + Y_{N-2}}{2\,h} + O(h^2) \tag{8.142}$$

Then the boundary condition (8.140) is approximated by

$$\frac{3\,y_N - 4\,y_{N-1} + y_{N-2}}{2\,h} + k\, y_N = g_2 \tag{8.143}$$

It can be shown that the resulting difference scheme is again second-order accurate. There are other possibilities to approximate the boundary condition (8.140) so that second-order accuracy of the overall difference scheme is maintained, e.g. the one employed in Section 9.3 to numerically treat the derivative initial condition through the use of artificial variables (cf. (9.32) and the discussion there).

PROBLEMS

1. In general, the study of the existence and uniqueness of a solution for boundary value problems is more complicated. Consider the boundary value problem

$$\begin{cases} Y'' = 0, & 0 < x < 1 \\ Y'(0) = g_1, & Y'(1) = g_2 \end{cases}$$

Show that the problem has no solution if $g_1 \neq g_2$, and infinite many solutions when $g_1 = g_2$.

Hint: For the case $g_1 \neq g_2$, integrate the differential equation over [0, 1].

2. As another example of solution nonuniqueness, verify that for any constant c, $Y(x) = c\,\sin(x)$ solves the boundary value problem

$$\begin{cases} Y''(x) + Y(x) = 0, & 0 < x < \pi \\ Y(0) = Y(\pi) = 0 \end{cases}$$