Part 1: Overview of Ordinary Differential Equations

Chapter 1

Basic Concepts and Problems

1.1 Problems Leading to Ordinary Differential Equations

Many scientific and engineering problems are modeled by systems of ordinary differential equations (ODEs). Some examples [3] follow.

1. *Mechanical Vibrations.* Let $y(t)$ denote the displacement at time t of a block of mass m that is connected to a spring of stiffness k, a damper of resistance c, and an oscillator $f(t)$ (Figure 1.1.1). If the system is released from position y_0 with a velocity y_0' then its subsequent motion satisfies

$$m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky = f(t), \qquad t > 0, \qquad y(0) = y_0, \qquad \frac{dy(0)}{dt} = y_0'.$$

   This is an example of an initial value problem (IVP) for a second-order linear ODE.

   - The variable $y(t)$ is called:
   - The variable t is called:
   - The ODE is *second order* because:
   - It's *linear* because:
   - This is an *IVP* because:

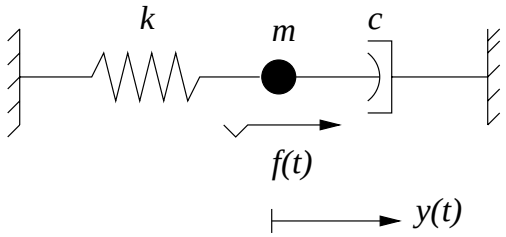

Figure 1.1.1: Motion of a spring-mass-damper system.

2. *Ecology.* Consider a population of predators and prey living in an ecological niche. The predators survive by eating the prey and the prey exist on an independent source of food. A classical situation involves foxes and rabbits. Let $P(t)$ and $p(t)$, respectively, denote the populations of predators and prey at time t. Given the initial populations P_0 and p_0 of predators and prey, their subsequent populations satisfy the *Lotka-Volterra* equations

$$\frac{dp}{dt} = p(a - \alpha P), \qquad t > 0, \qquad p(0) = p_0,$$
$$\frac{dP}{dt} = P(-c + \gamma p), \qquad t > 0, \qquad P(0) = P_0,$$

where a, c, α, and γ are positive constants corresponding to the prey's natural growth rate, the predator's natural death rate, the prey's death rate upon coming into contact with predators, and the predator's growth rate upon coming into contact with prey. This example involves an IVP for a system of two first-order nonlinear ODEs.

   - The ODEs are *nonlinear* because:
   - The system is *first order* because:

3. *Column Buckling.* The lateral displacement $y(t)$ at position t of a clamped-hinged bar of length l that is subjected to a load P (Figure 1.1.2) may be approximated by the *Euler-Bernoulli* equations

$$\frac{d^4y}{dt^4} + \lambda\frac{d^2y}{dt^2} = 0, \qquad 0 < t < l,$$

with the boundary conditions

$$y(0) = \frac{dy(0)}{dt} = 0, \qquad y(l) = \frac{d^2y(l)}{dt^2} = 0.$$

The parameter $\lambda = P/EI$, where EI is the flexural rigidity of the bar. This is an example of a boundary value problem (BVP) for a fourth-order linear ODE.

- This is a *boundary value* problem because:

Observe that $y(t) = 0$ is a solution of this problem for all values of λ and l. A more interesting problem is to determine y and those values of λ and l for which non-trivial ($y(t) \neq 0$) solutions exist. As such, this BVP is also a differential eigenvalue problem. Nontrivial solutions $y(t)$ are called *eigen functions* and their corresponding values of λ are *eigenvalues*.

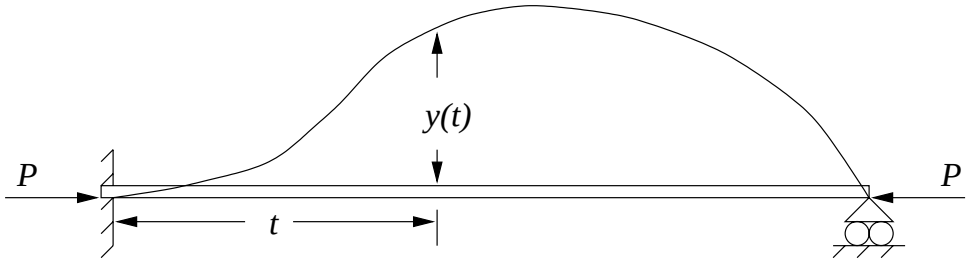

Figure 1.1.2: Buckling of an elastic column.

4. *Pendelum Oscillations* ([1], Section 1.3). The position $(x(t), y(t))$ at time t of a particle of mass m oscillating on a pendelum of length l (Figure 1.1.3) is

$$m\frac{d^2x}{dt^2} = -T\sin\theta = -\frac{T}{l}x, \qquad m\frac{d^2y}{dt^2} = mg - T\cos\theta = mg - \frac{T}{l}y, \qquad t > 0,$$

where g is the acceleration of gravity, $T(t)$ is the tension in the string, and $\theta(t)$ is the angle of the pendelum relative to the vertical at time t (Figure 1.1.3). These equations are, however, insufficient to guarantee that the particle stays on the string. To ensure that this is so, we must supplement the ODEs by the algebraic constraint

$$x^2 + y^2 = l^2.$$

The two second-order differential equations and the constraint comprise a system of five *differential algebraic equations* (DAEs) for the unknowns $x(t)$, $y(t)$, and $T(t)$. Initial conditions specify $x(0)$, $dx(0)/dt$, $y(0)$, and $dy(0)/dt$, but not $T(0)$.

Of course, for this problem, it's easy to eliminate the constraint by introducing the change of variables $x = l\sin\theta$, $y = l\cos\theta$. This would reduce the DAEs to the second-order ODE

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta.$$

Such simplifications would not be possible with more difficult systems.

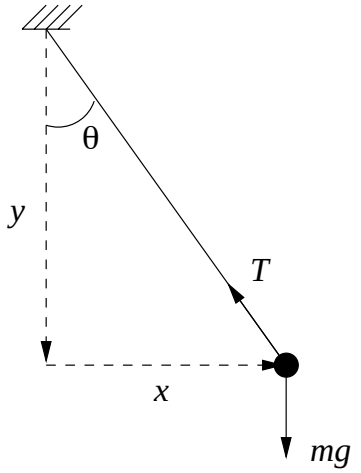

Figure 1.1.3: Oscillations of a simple Pendelum.

IVPs will comprise our initial study (Part 2 of these notes). We'll take up BVPs next (Part 3) and conclude with a study of DAEs (Part 4). In almost all cases, it will suffice to develop and analyze numerical methods for IVPs and BVPs that are written as first-order vector systems of ODEs in the *explicit* form

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \qquad t > 0, \tag{1.1.1a}$$

where

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix}, \qquad \mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} f_1(t, y_1, \ldots, y_m) \\ f_2(t, y_1, \ldots, y_m) \\ \vdots \\ f_m(t, y_1, \ldots, y_m) \end{bmatrix}, \tag{1.1.1b}$$

and $\mathbf{y}' := d\mathbf{y}/dt$. Data for IVPs would is specified as

$$\mathbf{y}(0) = \mathbf{y}_0 = \begin{bmatrix} y_{10} \\ y_{20} \\ \vdots \\ y_{m0} \end{bmatrix}. \tag{1.1.1c}$$

Boundary data is more complex. The most general boundary conditions specify a nonlinear relationship between $\mathbf{y}(0)$ and $\mathbf{y}(l)$ of the form

$$\mathbf{g}(\mathbf{y}(0), \mathbf{y}(l)) = \begin{bmatrix} g_1(\mathbf{y}(0), \mathbf{y}(l)) \\ g_2(\mathbf{y}(0), \mathbf{y}(l)) \\ \vdots \\ g_m(\mathbf{y}(0), \mathbf{y}(l)) \end{bmatrix} = \mathbf{0}. \tag{1.1.1d}$$

In many cases the extension of a scalar to a vector IVP will be obvious and we can study methods for a scalar IVP

$$y'(t) = f(t, y), \qquad t > 0, \qquad y(0) = y_0, \tag{1.1.2}$$

Example 1.1.1. Higher-order ODEs can be written as first-order systems. Consider the $m\ th$ order equation

$$z^{(m)} = g(t, z, z', \ldots, z^{(m-1)}),$$

where $z^{(i)} := d^i z/dt^i$, and introduce the new variables

$$y_1 = z, \qquad y_2 = z', \qquad y_3 = z'', \qquad \ldots, \qquad y_m = z^{(m-1)}.$$

Then, we have a system in the form (1.1.1) with

$$y_1' = y_2, \qquad y_2' = y_3, \qquad \ldots, \qquad y_m' = g(t, y_1, y_2, \ldots, y_m).$$

For an IVP, that data would prescribe as

$$z^{(i)}(0) = c_i, \qquad i = 0, 1, \ldots, m-1,$$

with the understanding that $z^{(0)} = z$. Written in terms of $\mathbf{y}$, we have

$$y_1(0) = c_0, \qquad y_2(0) = c_1, \qquad \ldots \qquad y_m(0) = c_{m-1}.$$

With DAEs and sometimes with IVPs and BVPs, it will be necessary to consider *implicit* differential systems

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}. \tag{1.1.3}$$

If the Jacobian $\partial\mathbf{F}/\partial\mathbf{y}'$ is nonsingular then, by the implicit function theorem, (1.1.3) is equivalent to (1.1.1); however, it may be natural to solve (1.1.3) in its implicit form to maintain, e.g., sparsity. DAEs involve systems where $\partial\mathbf{F}/\partial\mathbf{y}'$ is singular.

Example 1.1.2. Let us write the oscillating pendelum problem as a first-order system by letting

$$y_1 = x, \qquad y_2 = x', \qquad y_3 = y, \qquad y_4 = y'.$$

Then, we have

$$y_1' = y_2, \qquad y_2' = -\frac{T}{ml}y_1, \qquad y_3' = y_4, \qquad y_4' = g - \frac{T}{ml}y_3, \qquad y_5 = T$$

with the constraint

$$y_1^2 + y_3^2 = l^2.$$

This can be written in the form (1.1.3) with

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = \begin{bmatrix} y_1' - y_2 \\ y_2' + \frac{y_1 y_5}{ml} \\ y_3' - y_4 \\ y_4' - g + \frac{y_3 y_5}{ml} \\ y_1^2 + y_3^2 - l^2 \end{bmatrix} = \mathbf{0}.$$

The Jacobian

$$\mathbf{F}_\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

is clearly singular.

Problems

1. Letting $y_1 = p$ and $y_2 = P$, write the predator-prey model

$$\frac{dp}{dt} = p(a - \alpha P), \qquad t > 0, \qquad p(0) = p_0,$$

$$\frac{dP}{dt} = P(-c + \gamma p), \qquad t > 0, \qquad P(0) = P_0,$$

   in the vector form (1.1.1).

2. Write the mechanical vibration problem

$$m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky = f(t), \qquad t > 0, \qquad y(0) = y_0, \qquad \frac{dy(0)}{dt} = y_0',$$

   in the form (1.1.1a-c)

3. Write the column buckling problem

$$\frac{d^4y}{dt^4} + \lambda \frac{d^2y}{dt^2} = 0, \qquad 0 < t < l, \qquad y(0) = \frac{dy(0)}{dt} = 0, \qquad y(l) = \frac{d^2y(l)}{dt^2} = 0,$$

in the form (1.1.1a,b,d)

1.2 Existence, Uniqueness, and Stability of IVPs

Before considering numerical procedures for (1.1.1, 1.1.3), let us review some results from ODE theory that ensure the existence of unique solutions which depend continuously on the initial data. For simplicity, let us focus on a scalar IVP having the form (1.1.2).

Definition 1.2.1. *A function $f(t, y)$ satisfies a Lipschitz condition in a domain D if there exists a non-negative constant L such that*

$$|f(t, y) - f(t, z)| \leq L|y - z|, \qquad \forall (t, y), (t, z) \in D. \tag{1.2.4}$$

Satisfying a Lipschitz condition guarantees that solutions $y(t)$ of (1.1.2) are unique as expressed by the following theorem.

Theorem 1.2.1. *Let $f(t, y)$ be continuous and satisfy a Lipschitz condition on*

$$\{(t, y) \mid 0 \leq t \leq T, -\infty < y < \infty\}.$$

Then the IVP (1.1.2) has a unique continuously differentiable solution on $[0, T]$ for all $y_0 \in (-\infty, \infty)$.

Proof. The proof appears in most books on ODE theory, e.g., [2]. □

Remark 1. A Lipschitz conditions guarantees unique solutions and is not needed for esistence. For example, the IVP

$$y' = 3y^{2/3}, \qquad y(0) = 0,$$

has the two solution $y(t) = 0$ and $y(t) = t^3$. This $f(t, y) = 3y^{2/3}$ does not satisfy a Lipschitz condition.

Remark 2. Theorem 1.2.1 applies when f satisfys a Lipschitz condition on a compact, rather than an unbounded, domain D as long as the solution $y(t)$ remains in D [2].

In addition to existence and uniqueness, we will want to know something about the stability of solutions of the IVP. In particular, we will usually be interested in the sensitivity of the solution to small changes in the data. Perturbations arise naturally in numerical computation due to discretization and round off errors. A formal study of sensitivity would lead us to the following notion of a well-posed system.

Definition 1.2.2. *The IVP (1.1.2) is well-posed if there exists positive constants k and $\hat{\epsilon}$ such that, for any $\epsilon \leq \hat{\epsilon}$, the perturbed IVP*

$$z' = f(t,z) + \delta(t), \qquad z(0) = y_0 + \epsilon_0, \tag{1.2.5}$$

satisfies

$$|y(t) - z(t)| \leq k\epsilon \tag{1.2.6}$$

whenever $|\epsilon_0| < \epsilon$ and $|\delta(t)| < \epsilon$ for $t \in [0,T]$.

Again, a Lipschitz condition ensures that we are dealing with a well-posed IVP.

Theorem 1.2.2. *If $f(t,y)$ satisfies a Lipschitz condition on*

$$\{(t,y) \mid 0 \leq t \leq T, -\infty < y < \infty\}$$

then $y' = f(t,y)$ is well posed on $[0,T]$ with respect to all initial data.

Proof. Let

$$\zeta(t) = z(t) - y(t)$$

and subtract (1.1.2) from (1.2.5) to obtain

$$\zeta'(t) = f(t,z) - f(t,y) + \delta(t), \qquad \zeta(0) = \epsilon_0.$$

Taking an absolute value and using the Lipschitz condition (1.2.4)

$$|\zeta'(t)| \leq L|\zeta(t)| + |\delta(t)|, \qquad |\zeta(0)| = \epsilon_0.$$

We easily see that $|\zeta(t)|' \leq |\zeta'(t)|$ provided that $|\zeta(t)|'$ exists and it is fairly easy to show that $|\zeta(t)|'$ exists. Additionally, by assumption,

$$\max_{0\leq t\leq T} |\delta(t)| < \epsilon, \qquad \max_{0\leq t\leq T} |\epsilon_0| < \epsilon,$$

so

$$|\zeta(t)|' \leq L|\zeta(t)| + \epsilon, \qquad |\zeta(0)| < \epsilon.$$

Multiply by the integrating factor e^{-Lt} to obtain

$$(e^{-Lt}|\zeta(t)|)' \leq \epsilon e^{-Lt}, \qquad |\zeta(0)| < \epsilon.$$

Integrating

$$|\zeta(t)| \leq \frac{\epsilon}{L[(L+1)e^{Lt} - 1]}.$$

Since the denominator is smallest when $t = 0$

$$|z(t) - y(t)| \leq \frac{\epsilon}{L^2} = k\epsilon, \qquad \forall t \in [0, T];$$

thus, $y' = f(t, y)$ is well posed on $[0, T]$. □

The notion of a well-posed system is related to the more common notion of stability as indicated by the following definition.

Definition 1.2.3. *Consider the differential equation $y' = f(t, y)$ and (without loss of generality) let the origin $(0, 0)$ be an equilibrium point, i.e., $f(0, 0) = 0$. Then, the origin is:*

- *stable if a perturbation of the initial condition $|y(0)| < \epsilon$ grows no larger than ϵ for subsequent times, i.e., if $|y(t)| < \epsilon$ for $t > 0$.*
- *asymptotically stable if it is stable and $|y(0)| < \epsilon$ implies that $\lim_{t\to\infty} |y(t)| = 0$.*
- *unstable if it is not stable.*

Remark 3. This definition could, like Definition 1.2.2, also involve perturbations of $f(t, y)$. We have omitted these for simplicity.

An autonomous system is one where $f(t, y)$ does not explicitly depend on t, i.e., $f(t, y) = f(y)$. If $(0, 0)$ is an equilibrium point then, in this case, $f(0) = 0$. Expanding the solution in a Taylor's series in y, we have

$$y'(t) = f(y) = f(0) + f_y(0)y(t) + O(y^2).$$

Since $f(0) = 0$,

$$y'(t) = f_y(0)y(t) + O(y^2),$$

where $f_y = \partial f/\partial y$. Recall that a function $g(y)$ is $O(y^p)$ if there exists a constant $C > 0$ such that $|g(y)| \leq Cy^p$ as $y \to 0$.

Letting $\lambda = f_y(0)$, we see that the stability of an autonomous system is related to that of the simple linear IVP

$$y' = \lambda y, \qquad t > 0, \qquad y(0) = \epsilon. \tag{1.2.7}$$

Additional details on autonomous and non-autonomous systems appear in Birkhoff and Rota [2] or Hairer *et al.* [5], Section 1.13. The solution of the linear IVP (1.2.7) is

$$y(t) = \epsilon e^{\lambda t};$$

hence, (1.2.7) is

- stable when $Re(\lambda) \leq 0$,
- asymptotically stable when $Re(\lambda) < 0$, and
- unstable when $Re(\lambda) > 0$.

These conclusions remain true for the original nonlinear autonomous problem when $Re(\lambda) \neq 0$ and y is small enough for the $O(y^2)$ term to be negligible relative to λy. This cannot happen in the stable case ($Re(\lambda) = 0$); hence, it requires a more careful analysis.

Remark 4. The analysis of non-autonomous systems is similar [1].

Analyzing the stability of autonomous vector systems

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}(t)) \tag{1.2.8}$$

is more complicated. Once again, assume that $\mathbf{y} = \mathbf{0}$ is an equilibrium point and expand $\mathbf{f}(\mathbf{0})$ in a series

$$\mathbf{y}' = \mathbf{f}(\mathbf{0}) + \mathbf{f}_\mathbf{y}(\mathbf{0})\mathbf{y} + O(\|\mathbf{y}\|^2) = \mathbf{f}_\mathbf{y}(\mathbf{0})\mathbf{y} + O(\|\mathbf{y}\|^2).$$

We need a brief digression for a few definitions.

Definition 1.2.4. *The Jacobian matrix of a vector-valued function* $\mathbf{f}(\mathbf{y})$ *with respect to* $\mathbf{y}$ *is the matrix*

$$\mathbf{f}_\mathbf{y} := \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & & \frac{\partial f_1}{\partial y_m} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & & \frac{\partial f_2}{\partial y_m} \\ & & \ddots & \\ \frac{\partial f_m}{\partial y_1} & \frac{\partial f_m}{\partial y_2} & & \frac{\partial f_m}{\partial y_m} \end{bmatrix}. \tag{1.2.9}$$

Definition 1.2.5. *The norm of a vector* $\mathbf{y}$ *is a scalar* $\|\mathbf{y}\|$ *such that*

1. $\|\mathbf{y}\| \geq 0$ *and* $\|\mathbf{y}\| = 0$ *if and only if* $\mathbf{y} = \mathbf{0}$,
2. $\|\alpha\mathbf{y}\| = |\alpha|\|\mathbf{y}\|$ *for any scalar* α, *and*
3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

We'll identify specific vector norms when they are needed. For the moment, let's return to our stability analysis and let $\mathbf{A} = \mathbf{f}_\mathbf{y}(\mathbf{0})$. Then, the stability of the nonlinear autonomous system (1.2.8) is related to that of the linear system

$$\mathbf{y}' = \mathbf{A}\mathbf{y}, \qquad \mathbf{y}(0) = \mathbf{y}_0, \qquad \|\mathbf{y}_0\| \leq \epsilon. \tag{1.2.10a}$$

The solution of this system is

$$\mathbf{y}(t) = e^{\mathbf{A}t}\mathbf{y}_0, \tag{1.2.10b}$$

where the matrix exponential is defined by the series

$$e^{\mathbf{A}t} = \mathbf{I} + t\mathbf{A} + \frac{t^2}{2!}\mathbf{A}^2 + \frac{t^3}{3!}\mathbf{A}^3 + \cdots. \tag{1.2.10c}$$

Often, $\mathbf{A}$ can be diagonalized as

$$\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_m \end{bmatrix} \tag{1.2.11}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_m$ are the eigenvalues of $\mathbf{A}$ and the columns of $\mathbf{T}$ are the corresponding eigenvectors. In this case, we may easily verify that the solution of (1.2.10) is

$$\mathbf{y}(t) = \mathbf{T}e^{\mathbf{\Lambda}t}\mathbf{T}^{-1}\mathbf{y}_0.$$

Thus, (1.2.10) is stable if all of the eigenvalues have non-positive real parts, asymptotically stable if all of the eigenvalues have negative real parts, and unstable otherwise.

Unfortunately, not all matrices are diagonalizable. However, $\mathbf{A}$ can always be reduced to the Jordan canonical form

$$\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \begin{bmatrix} \mathbf{\Lambda}_1 & & & \\ & \mathbf{\Lambda}_2 & & \\ & & \ddots & \\ & & & \mathbf{\Lambda}_l \end{bmatrix} \tag{1.2.12a}$$

where each Jordan block has the form

$$\mathbf{\Lambda}_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix} \tag{1.2.12b}$$

The dimension of the Jordan block $\mathbf{\Lambda}_i$ corresponds to the multiplicity of the eigenvalue λ_i. Thus, if λ_i is simple, the block is a scalar. With this, it is relatively easy to show [2] that $\mathbf{y} = \mathbf{0}$ is

- stable when either $Re(\lambda_i) < 0$ or $Re(\lambda_i) = 0$ and λ_i is simple, $i = 1, 2, \ldots, m$,
- asymptotically stable when $Re(\lambda_i) < 0$, $i = 1, 2, \ldots, m$, and
- unstable otherwise.

Example 1.2.1. Consider the predator-prey model of Section 1.1

$$y_1' = y_1(a - \alpha y_2), \qquad y_2' = y_2(-c + \gamma y_1),$$

where a, α, c, and γ are positive constants. This autonomous problem has two equilibrium points:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} c/\gamma \\ a/\alpha \end{bmatrix}.$$

The point $[0,0]^T$ corresponds to extinction of predators and prey, whereas $[c/\gamma, a/\alpha]^T$ implies a co-existence of both species. The stability of $[0,0]^T$ may be analyzed by using the linear system

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} a & 0 \\ 0 & -c \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

Since the system matrix is diagonal, its eigenvalues are $\lambda_1 = a$, $\lambda_2 = -c$. Thus, $\lambda_1 > 0$ and $[0,0]^T$ is unstable.

In order to analyze the stability of the second equilibrium point, we introduce the change of variables

$$z_1 = y_1 - \frac{c}{\gamma}, \qquad z_2 = y_2 - \frac{a}{\alpha}$$

to move the equilibrium point to the origin. Substituting this transformation into the predator-prey equations gives

$$z_1' = (z_1 + \frac{c}{\gamma})[a - \alpha(z_2 + \frac{a}{\alpha})] = -\alpha z_2(\frac{c}{\gamma} + z_1),$$
$$z_2' = (z_2 + \frac{a}{\alpha})[-c + \gamma(z_1 + \frac{c}{\gamma})] = \gamma z_1(\frac{a}{\alpha} + z_2).$$

The linearized system is now

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}' = \begin{bmatrix} 0 & -\alpha c/\gamma \\ a\gamma/\alpha & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}.$$

The eigenvalues of this system are $\lambda_{1,2} = \pm i\sqrt{c/a}$. Since the eigenvalues are imaginary, the linear system is stable; however, the stability of the nonlinear system is undetermined without additional analysis.

Problems

1. Suppose $\partial f/\partial y$ is continuous on a closed convex domain D. Show that $f(t,y)$ satisfies a Lipschitz condition on D with

   $$L = \max_{(t,y)\in D} |\frac{\partial f(t,y)}{\partial y}|.$$

   (Hint: use the Mean Value Theorem.)

2. Are the following IVPs well posed ([4], p. 23)? Explain.

   $$y' = \sqrt{1-y^2}, \qquad y(0) = 0,$$
   $$y' = \sqrt{y^2-1}, \qquad y(0) = 2.$$

Bibliography

[1] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[2] G. Birkhoff and G.-C. Rota. *Ordinary Differential Equations.* John Wiley and Sons, New York, third edition, 1978.

[3] W.E. Boyce and R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley and Sons, New York, third edition, 1977.

[4] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice Hall, Englewood Cliffs, 1971.

[5] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, Berlin, second edition, 1993.

Part 2: Initial Value Problems

Chapter 2

Introduction

2.1 The Explicit Euler Method: Convergence and Stability

As an introductory example, we examine, perhaps, the simplest method for solving the first-order scalar IVP

$$y' = f(t, y), \qquad t > 0, \qquad y(0) = y_0. \tag{2.1.1}$$

This method is called *Euler's method* (the *Euler-Cauchy*, *forward Euler*, or *explicit Euler* method). Euler's method and the scalar IVP (2.1.1) illustrate the fundamental concepts of consistency, convergence, and stability without requiring undo complexity.

We see that $y(0)$ and $y'(0) = f(0, y(0))$ may be evaluated from the prescribed initial data. Thus, we can determine y for a small value of t, say $t = h$, by expanding y in a Taylor's series about $t = 0$ and retaining only two terms. Using (2.1.1)

$$y(h) = y(0) + hy'(0) + O(h^2) = y_0 + hf(0, y_0) + O(h^2).$$

Euler's method is obtained by neglecting the $O(h^2)$ remainder terms. We'll call the answer that we get upon neglecting these terms y_1 in order to distinguish it from the exact value $y(h)$. Thus, we have

$$y_1 = y_0 + hf(0, y_0).$$

The process can be repeated by calculating a second Taylor's series about $t = h$ to obtain the approximate solution at $t = 2h$

$$y_2 = y_1 + hf(h, y_1).$$

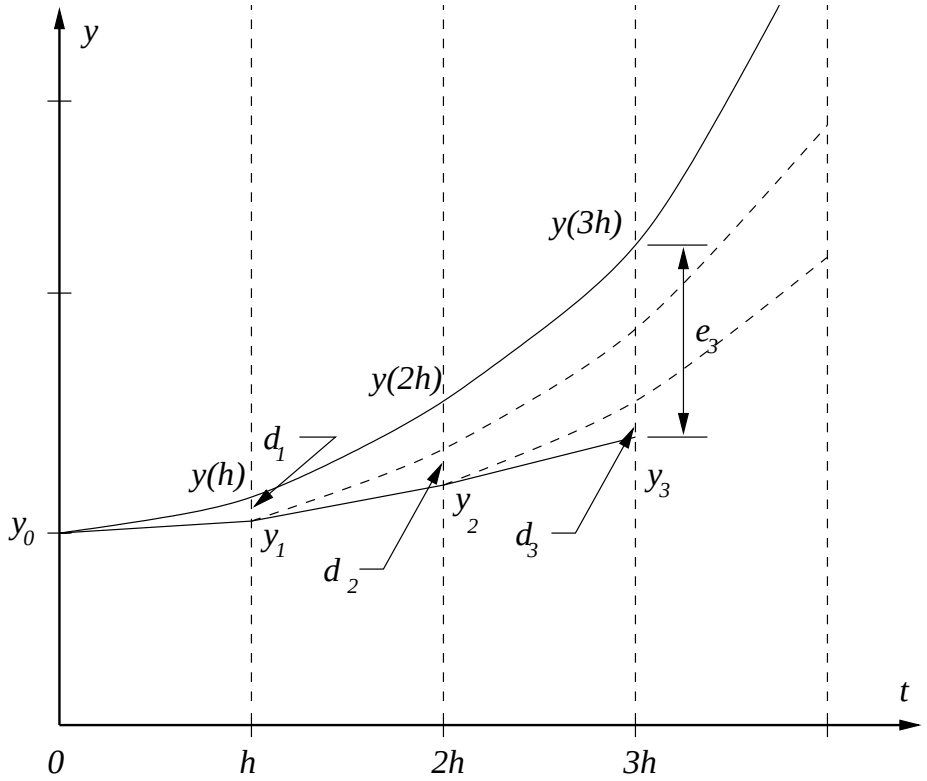

Figure 2.1.1: Exact solution and Euler approximation of $y' = f(t, y)$ with local and global errors shown.

In general, the approximate solution at $t = (n + 1)h$, obtained by an expansion about $t = nh$, is

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}), \qquad t_n = nh, \qquad n > 0. \tag{2.1.2}$$

Once again, y_n denotes the approximation of the exact solution $y(t_n)$ of (2.1.1) at $t = t_n = nh$. Because we have retained only two terms of the Taylor's expansion and because h can be regarded as being a continuous variable ranging from, e.g., t_{n-1} to t_n, we can interpret the Euler solution (2.1.2) as being a straight line extrapolation of the exact solution passing through the point (t_{n-1}, y_{n-1}) (Figure 2.1.1). Thus, the Euler solution y_n is on a line tangent to the exact solution of the ODE that passes through (t_{n-1}, y_{n-1}).

Errors are introduced at each step of the calculation. Among other things, we'll want to know that these errors decrease with decreasing step size h. This analysis will be

simplified by defining two errors: local errors d_n that are introduced in any single step of the computation and global errors e_n that include the accumulation of local errors over several steps. Both will subsequently be defined more precisely; however, for the moment, we show each in Figure 2.1.1.

Example 2.1.1. Consider the very simple IVP

$$y' = \lambda y, \qquad y(0) = 1,$$

which, of course, has the exact solution

$$y(t) = e^{\lambda t}.$$

For this problem, $f(t, y) = \lambda y$; thus, Euler's method (2.1.2) is

$$y_n = (1 + h\lambda)y_{n-1}, \qquad n > 0, \qquad y_0 = 1.$$

With $\lambda = -100$ and $h = 0.001$, we have

$$y_n = 0.9y_{n-1}, \qquad n > 0, \qquad y_0 = 1.$$

Results for the computed solution and global error

$$e_n = y(t_n) - y_n \tag{2.1.3}$$

are presented in Table 2.1.1 for $n \in [0, 10]$. The error, as may be expected, grows with the number of steps. Results in Table 2.1.2 display the solution and global error at $t = 0.1$ for $\lambda = -100$ and several choices of h. It appears that e_n is decreasing linearly with h.

Let us verify that the results of Example 2.1.1 hold more generally. We will need definitions of the various errors prior to establishing a result.

Definition 2.1.1. *The local error is the amount by which the numerical and exact solutions differ after one step assuming that both were exact at and prior to the beginning of the step.*

n	t_n	y_n	e_n
0	0.000	1.00000	0.00000
1	0.001	0.90000	0.00484
2	0.002	0.81000	0.00873
3	0.003	0.72900	0.01182
4	0.004	0.65610	0.01422
5	0.005	0.59049	0.01604
6	0.006	0.53144	0.01737
7	0.007	0.47830	0.01829
8	0.008	0.43046	0.01886
9	0.009	0.38742	0.01915
10	0.010	0.34868	0.01920

Table 2.1.1: The solution and error of $y' = \lambda y$, $y(0) = 1$, obtained by Euler's method with $\lambda = -100$ and $h = 0.001$.

n	$h = T/n$	y_n	$e_n/e^{\lambda T}$
10^2	10^{-3}	$2.66 \cdot 10^{-5}$	$4.15 \cdot 10^{-1}$
10^3	10^{-4}	$4.32 \cdot 10^{-5}$	$4.91 \cdot 10^{-2}$
10^4	10^{-5}	$4.52 \cdot 10^{-5}$	$4.99 \cdot 10^{-3}$

Table 2.1.2: The solution and error of $y' = \lambda y$, $y(0) = 1$, at $T = 0.1$ obtained by Euler's method with $\lambda = -100$ and $h = 10^{-3}$, 10^{-4}, 10^{-5}.

Example 2.1.2. According to Definition 2.1.1, the local error d_n at t_n is

$$d_n = y(t_n) - y_n \tag{2.1.4a}$$

assuming that $y_k = y(t_k)$, $k = n-1, n-2, \ldots, 0$. For Euler's method (2.1.2),

$$d_n = y(t_n) - y(t_{n-1}) - hf(t_{n-1}, y(t_{n-1})). \tag{2.1.4b}$$

Expanding $y(t_n)$ in a Taylor's series about t_{n-1}

$$y(t_n) = y(t_{n-1}) + hy'(t_{n-1}) + \frac{h^2}{2}y''(\xi_n), \qquad t_{n-1} < \xi_n < t_n.$$

Using (2.1.1) to eliminate $y'(t_{n-1})$ and substituting the result into (2.1.4b) yields

$$d_n = \frac{h^2}{2}y''(\xi_n), \qquad t_{n-1} < \xi_n < t_n. \tag{2.1.4c}$$

The local error for Euler's method is illustrated in Figure 2.1.1.

The local discretization error or local trunctaion error is closely related to the local error. Before defining it, let us write the difference equation (2.1.2) in a form that more closely resembles the ODE. Thus, we define the difference operator

$$\mathcal{L}_h u(t_n) := \frac{u(t_n) - u(t_{n-1})}{h} - f(t_{n-1}, u(t_{n-1})). \tag{2.1.5a}$$

Definition 2.1.2. *The local discretization error or local truncation error is the amount by which the exact solution of the ODE fails to satisfy the difference operator.*

The local discretization error for Euler's method (2.1.2) is

$$\tau_n = \mathcal{L}_h y(t_n). \tag{2.1.5b}$$

Comparing this result with (2.1.4b), reveals that $\tau_n = d_n/h = (h/2)y''(\xi_n)$; however, we shall see that the local and local discretization errors can have a more complex relationship for other difference methods.

Definition 2.1.3. *A difference method is consistent to order p if $\tau_n = O(h^p)$. If $p \geq 1$, the difference scheme is said to be consistent.*

Thus, Euler's method (2.1.2) is consistent of order one or, simply, consistent.

A numerical method converges if its global discretization error (2.1.3) approaches zero as the mesh is refined.

Definition 2.1.4. *Consider a calculation performed on $0 < t \leq T$ with $h = T/N$ and $n = 1, 2, \ldots, N$. A numerical method is convergent if*

$$\lim_{N\to\infty, h\to 0, Nh=T} |e_n| = 0, \qquad \forall n \in [0, N]. \tag{2.1.6}$$

If $e_n = O(h^p)$, the method is said to converge to order p.

The computations are performed on a sequence of meshes having finer-and-finer spacing, but always ending at the same time T. Although the definition has been stated with sequences of uniform meshes, it could easily be revised to include more general mesh sequences.

With these preliminaries behind us, we are in a position to state and prove our main result. Once again, a Lipschitz condition on $f(t, y)$ will come to our aid.

Theorem 2.1.1. *Suppose $y''(t)$ exists and $f(t,y)$ satisfies a Lipschitz condition on the strip $\{(t,y) \mid 0 \le t \le T, -\infty < y < \infty\}$. Then Euler's method (2.1.2) converges to the solution of the IVP (2.1.1).*

Proof. From (2.1.4b), the exact solution of the IVP (2.1.1) satisfies

$$y(t_n) = y(t_{n-1}) + hf(t_{n-1}, y(t_{n-1})) + d_n.$$

Subtracting this from (2.1.2) and using (2.1.3)

$$e_n = e_{n-1} + h[f(t_{n-1}, y(t_{n-1})) - f(t_{n-1}, y_{n-1})] + d_n. \tag{2.1.7}$$

Since $f(t,y)$ satisfies a Lipschitz condition,

$$|f(t_{n-1}, y(t_{n-1})) - f(t_{n-1}, y_{n-1})| \le L_n |y(t_{n-1}) - y_{n-1}| = L_n |e_{n-1}|.$$

Taking the absolute value of (2.1.7) and using the triangular inequality and the above Lipschitz condition yields

$$|e_n| \le (1 + hL_n)|e_{n-1}| + |d_n|. \tag{2.1.8}$$

Let

$$L = \max_{1 \le n \le N} L_n, \qquad d = \max_{1 \le n \le N} |d_n|,$$

and write (2.1.8) as

$$|e_n| \le (1 + hL)|e_{n-1}| + d.$$

Since the inequality holds for all n

$$|e_n| \le (1 + hL)[(1 + hL)|e_{n-2}| + d] + d = (1 + hL)^2 |e_{n-2}| + d[1 + (1 + hL)].$$

Continuing the iteration

$$|e_n| \le (1 + hL)^n |e_0| + d[1 + (1 + hL) + \ldots + (1 + hL)^{n-1}].$$

Using the formula for the sum of a geometric series

$$\sum_{k=0}^{n-1} (1 + hL)^k = \frac{(1 + hL)^n - 1}{hL}$$

yields

$$|e_n| \leq (1+hL)^n |e_0| + \frac{d}{hL}[(1+hL)^n - 1]. \tag{2.1.9}$$

Equation (2.1.9) can be written in a more revealing form by using the following Lemma, which is common throughout numerical analysis.

Lemma 2.1.1. *For all real z*

$$1 + z \leq e^z. \tag{2.1.10}$$

Proof. Using a Taylor's series expansion

$$1 + z + \frac{z^2}{2} e^{\xi} = e^z, \qquad \xi \in (0, z).$$

Neglecting the positive third term on the left leads to (2.1.10). □

Now, using (2.1.10) with $z = hL$,

$$(1+hL)^n \leq e^{hLn} \leq e^{LT},$$

since $nh = t_n \leq T$ for $n \leq N$. Using the above expression in (2.1.9)

$$|e_n| \leq e^{LT}|e_0| + \frac{d}{hL}(e^{LT} - 1). \tag{2.1.11}$$

Since $y''(t)$ exists for $t \in [0, T]$, Euler's method is consistent and, using (2.1.4c), we can bound on d as

$$d = \max_{0 \leq n \leq N} |d_n| = \frac{h^2}{2} \max_{0 \leq t \leq T} |y''(t)|. \tag{2.1.12}$$

Substituting (2.1.12) into (2.1.11)

$$|e_n| \leq e^{LT}|e_0| + \frac{h}{2L}(e^{LT} - 1) \max_{0 \leq t \leq T} |y''(t)|.$$

Roundoff error is neglected, so $e_0 = y(0) - y_0 = 0$, and we have

$$|e_n| \leq \frac{h}{2L}(e^{LT} - 1) \max_{0 \leq t \leq T} |y''(t)|. \tag{2.1.13}$$

Thus, $\lim_{h \to 0} |e_n| \to 0$ and Euler's method converges. □

Remark 1. The assumption that $y''(t)$ exists is not necessary. A proof without this assumption appears in Hairer *et al.* [2], Section 1.7. They also show that the Lipschitz condition on $f(t, y)$ need only be satisfied on a compact domain rather than an infinite strip. Gear [1], Section 1.3, presents a proof with the additional assumption that $f(t, y)$ satisfy a Lipschitz condition on t instead of requiring $y''(t)$ to be bounded.

Example 2.1.3. Consider the simple problem of Example 2.1.1

$$y' = \lambda y, \qquad y(0) = 1.$$

Since we know the exact solution to this problem, we may use (2.1.12) to calculate

$$d = \frac{h^2}{2} \max_{0 \le t \le T} |y''(t)| = \frac{(h\lambda)^2}{2}.$$

With $\lambda = -100$ and $h = 0.001$, we have $d = 0.005$. This value may be compared with the actual local error 0.00484 that appears in the second line of Table 2.1.1. (Computed local error values are not available for the other steps recorded Table 2.1.1. Why?) In this case, the bound d compares quite favorably with the exact local error. However, bear in mind that we have very precise data for this example, including an exact solution.

With $f(t, y) = \lambda y$, we have

$$|f(t, y) - f(t, z)| = |\lambda||y - z|.$$

Hence, we may take $L = |\lambda| = 100$ as a Lipschitz constant. Then, using (2.1.13) with $T = 0.01$ yields

$$|e_n| \le \frac{0.001}{200}(e^{100(0.01)} - 1)10^4 \approx 0.0859.$$

Comparing this bound and the actual error on the last line of Table 2.1.1, we see that the bound is about 4.5 times the actual error. Comparing formula (2.1.13) with the results of Table 2.1.2, reveals that both predict a linear convergence rate in h.

Example 2.1.4. Let's solve the problem of Examples 2.1.1 and 2.1.3 with a different choice of h. Thus, we'll keep $\lambda = -100$ and choose $h = 0.05$ to obtain the difference equation (Example 2.1.1)

$$y_n = (1 + h\lambda)y_{n-1} = -4y_{n-1}, \qquad n > 0, \qquad y_0 = 1.$$

Results, shown in Table 2.1.3, are very puzzling. As n (t) increases, the solution increases exponentially in magnitude while oscillating from time step-to-time step. The results are clearly not approximating the decaying exponential solution of $y' = -100y$.

n	t_n	y_n
0	0.00	1
1	0.05	-4
2	0.10	16
3	0.15	-64
4	0.20	256
5	0.25	-1024
6	0.30	4096
7	0.35	-16384
8	0.40	65536

Table 2.1.3: The solution of $y' = \lambda y$, $y(0) = 1$, by Euler's method for $\lambda = -100$ and $h = 0.05$.

The results of Example 2.1.4 are displaying a "numerical instability." Stability of a numerical method is analogous to "well posedness" of a differential equation. Thus, we would like to know whether or not small changes in, e.g., the initial data or $f(t, y)$ produce bounded changes in the solution of the difference scheme.

Definition 2.1.5. *Let y_n, $n > 0$, be the solution of a one-step numerical method with initial condition y_0 and let z_n be the solution of the same numerical method with a perturbed initial condition $y_0 + \delta_0$. The one-step method is stable if there exists positive constants $\hat{h}$ and k such that*

$$|y_n - z_n| \leq k\delta, \qquad \forall nh \leq T, \qquad h \in (0, \hat{h}), \tag{2.1.14}$$

whenever $|\delta_0| \leq \delta$.

Remark 2. A one-step method, like Euler's method, only requires information about the solution at t_{n-1} to compute a solution at t_n. We'll have to modify this definition when dealing with multi-step methods.

Remark 3. The definition is difficult to apply in practice. It is too dependent on $f(t, y)$.

Example 2.1.5. Although Definition 2.1.5 is difficult to apply, we can apply it to Euler's method. Consider the original and perturbed problems

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}),$$

$$z_n = z_{n-1} + hf(t_{n-1}, z_{n-1}), \qquad z_0 = y_0 + \delta_0.$$

Let

$$\delta_n = z_n - y_n, \qquad n \geq 0,$$

and subtract the above difference equations to obtain

$$\delta_n = \delta_{n-1} + h[f(t_{n-1}, z_{n-1}) - f(t_{n-1}, y_{n-1})].$$

Using the Lipschitz condition (1.2.4)

$$|\delta_n| \leq (1 + hL)|\delta_{n-1}|.$$

Iterating the inequality and using (2.1.10) leads us to

$$|\delta_n| \leq (1 + hL)^n |\delta_0| \leq e^{hLn}|\delta_0| \leq e^{LT}|\delta_0| \leq k\delta, \qquad \forall nh \leq T,$$

where $k = e^{LT}$ and $|\delta_0| \leq \delta$.

Thus, Euler's method is stable. So what's wrong with the results of Table 2.1.3? This difficulty points out additional shortcomings of Definition 2.1.5. First, it is applicable in the limit of small step size. Computations are performed with a prescribed step size and it is often difficult to determine if this step size is "small enough." Second, Definition 2.1.5 allows some growth of the solution for bounded times. In Example 2.1.5, $k > 1$. If the solution of the IVP is stable or asymptotically stable, we cannot tolerate *any* growth in the computed solution unless either L is small or computations are performed for very short time periods T. If the solution of the IVP is unstable, some growth of the perturbation is acceptable. The following concept of absolute stability will provide a more useful tool than that of Definition 2.1.5 when IVP solutions are not growing in time.

Definition 2.1.6. *A numerical method is absolutely stable for a step size h and a given ODE if a change in y_0 of size δ is no larger than δ for all subsequent time steps.*

Remark 4. In contrast to Definition 2.1.5, absolute stability is applied at a specific value of h rather than in the limit as $h \to 0$.

Definition 2.1.6, like Definition 2.1.5, still depends too heavily on the differential equation. In order to reduce this dependence, it is common to apply absolute stability to the "test equation"

$$y' = \lambda y, \qquad y(0) = y_0. \tag{2.1.15}$$

In Chapter 1, we saw that (2.1.15) was useful in deciding the stability of the nonlinear differential equation (2.1.1). We now seek to use it to infer the stability of a difference equation.

Definition 2.1.7. *The region of absolute stability of a difference equation is the set of all real non-negative values of h and complex values of λ for which it is absolutely stable when applied to the test equation (2.1.15).*

Example 2.1.6. Consider Euler's method (2.1.2) applied to (2.1.15)

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

$$z_n = (1 + h\lambda)z_{n-1}, \qquad z_0 = y_0 + \delta_0$$

where $|\delta_0| \leq \delta$. Subtracting the two difference equations and, once again, letting $\delta_n = z_n - y_n$ yields

$$\delta_n = (1 + h\lambda)\delta_{n-1}.$$

Since the difference equation is linear, the perturbed problem satisfies the original difference equation with the perturbation as its initial condition.

Iterating, we find the solution of the perturbed problem to be

$$\delta_n = (1 + h\lambda)^n \delta_0.$$

Thus, the initial perturbation will not grow beyond $|\delta_0|$ if

$$|1 + h\lambda| \leq 1. \tag{2.1.16}$$

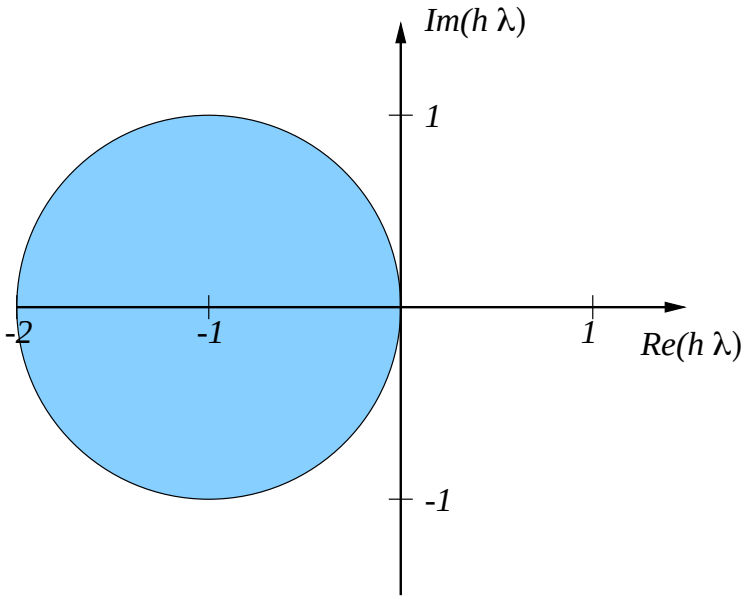

Figure 2.1.2: Region of absolute stability for Euler's method.

As shown in Figure 2.1.2, the region of absolute stability is a unit circle centered at $(-1, 0)$ in the complex $h\lambda$ plane.

Example 2.1.7. Let us solve the IVP

$$y' = y, \qquad y(0) = 1, \qquad 0 \le t \le 1$$

by Euler's method. The exact solution is, of course, $y(t) = e^t$. The interval $0 \le t \le 1$ is divided into $N = 2^k$, $k = 0, 1, \ldots$, uniform subintervals of width $h = 1/N$. In order to enhance roundoff effects, arithmetic was done on a simulated computer where floating point numbers have 21-bit fractions. The computed solutions and errors at $T = 1$ are presented in Table 2.1.4 for k ranging from 0 to 16. A ˜ signifies results computed with 21-bit rounded arithmetic.

Roundoff errors are introduced at each step of the computation because of the imprecise addition, multiplication, and evaluation of $f(t, y)$. They can, and typically do, accumulate to limit accuracy. In the present case, approximately four digits of accuracy can be achieved with a step size of approximately 2^{-14}. Decreasing the step size further will not increase accuracy. In order to study this, let

- $y(t_n)$ be the solution of the IVP at $t = t_n$,

k	$N = 2^k$	$h = 1/N$	$\tilde{y}_N$	$\tilde{e}_N$
0	1	1.00000000	2.00000000	-0.71828181
1	2	0.50000000	2.25000000	-0.46828184
2	4	0.25000000	2.44140625	-0.27687559
3	8	0.12500000	2.56578445	-0.15249738
4	16	0.06250000	2.63792896	-0.08035287
5	32	0.03125000	2.67698956	-0.04129227
6	64	0.01562500	2.69734669	-0.02093514
7	128	0.00781250	2.70773602	-0.01054581
8	256	0.00390625	2.71297836	-0.00530347
9	512	0.00195312	2.71561337	-0.00266846
10	1024	0.00097656	2.71694279	-0.00133904
11	2048	0.00048828	2.71764278	-0.00063904
12	4096	0.00024414	2.71795559	-0.00032624
13	8192	0.00012207	2.71811104	-0.00017079
14	16384	0.00006104	2.71814919	-0.00013264
15	32768	0.00003052	2.71804428	-0.00023755
16	65536	0.00001526	2.71732903	-0.00095280

Table 2.1.4: Solutions of $y' = y$, $y(0) = 1$, at $t = 1$ obtained by Euler's method with 21-bit rounded arithmetic.

- y_n be the infinite-precision solution of the difference equation at $t = t_n$, and
- $\tilde{y}_n$ be the computed solution of the difference equation at $t = t_n$.

The total error $\tilde{e}_n$ may be written as

$$|\tilde{e}_n| = |y(t_n) - \tilde{y}_n| \le |y(t_n) - y_n| + |y_n - \tilde{y}_n|. \tag{2.1.17a}$$

As usual, let

$$e_n = y(t_n) - y_n \tag{2.1.17b}$$

and also let

$$r_n = y_n - \tilde{y}_n. \tag{2.1.17c}$$

Thus,

$$|\tilde{e}_n| \le |e_n| + |r_n|. \tag{2.1.17d}$$

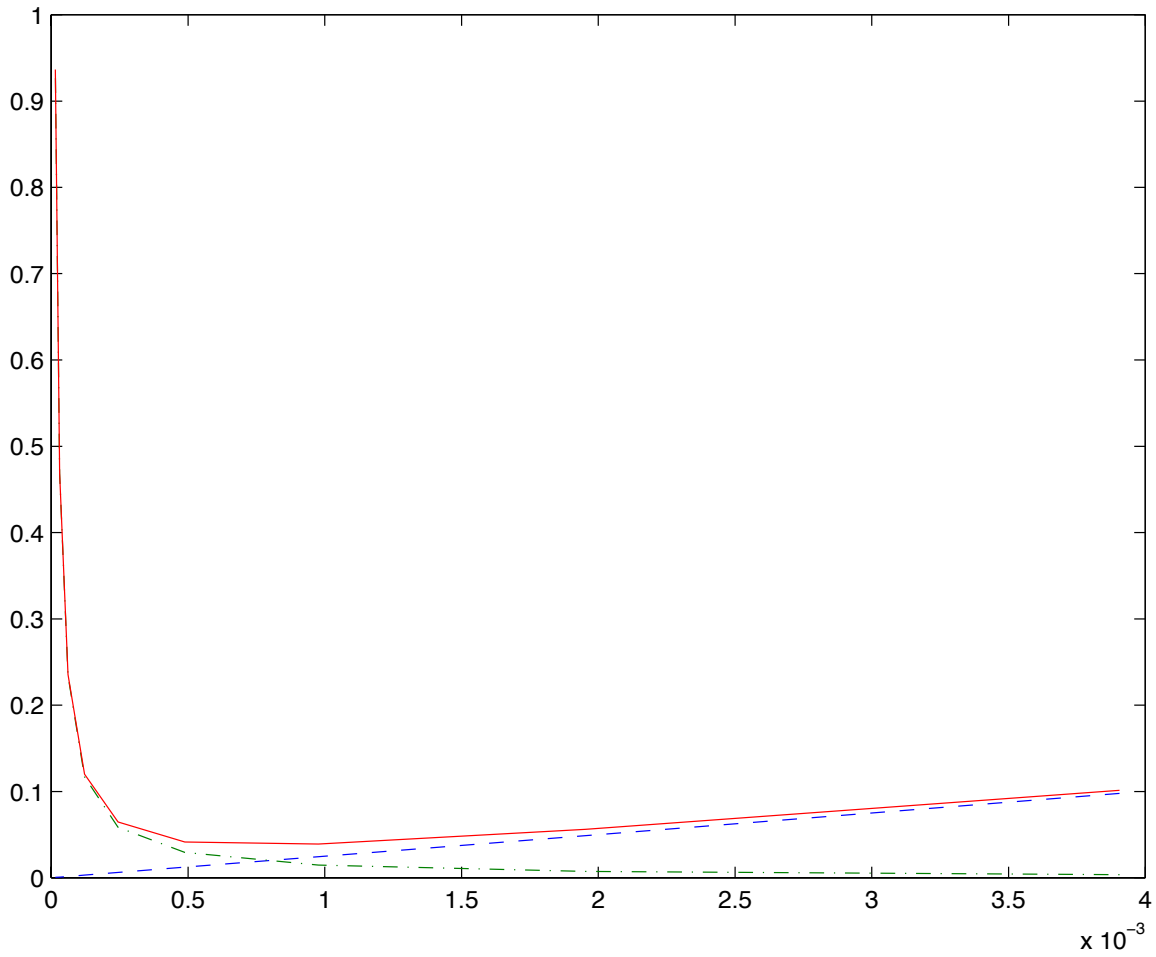

Figure 2.1.3: Total error $\tilde{e}_n$ (solid), discretization error e_n (dashed), and roundoff error r_n (dash-dot) as functions of step size h.

As previously noted, e_n is the global discretization error. We'll call r_n the round off error. According to Theorem 2.1.1, $|e_n| \leq kh$. An analysis similar to the one used in Theorem 2.1.1 reveals that $|r_n| \leq K/h$ ([1], pp. 21-23). Thus, while decreasing h decreases the discretization error it increases the bound on the roundoff error. As shown in Figure 2.1.3, there is an optimal value of h, h_{OPT}, that produces the minimum bound on the total error. Fortunately, roundoff error accumulation is not typically a problem when solving ODEs on realistic computers with the practical numerical methods that we shall study in subsequent sections.

Problems

1. Error bounds of the form (2.1.13) are called *a priori estimates* because they are

obtained without using the computed solution. Such *a priori* estimates are often very conservative. Indeed, we used relatively precise information to get the a priori error bound of Example 2.1.3. We can expect less precision when confronted with more realistic problems.

Error estimates that rely on the computed solution are called *a posteriori estimates*. Can you design an *a posteriori* procedure for estimating either the local or global errors of Euler's method? (Hint: you could try comparing solutions computed on different meshes.)

2. Consider a linear ODE with variable coefficients

$$y' = a(t)y + b(t)$$

Consider IVPs with initial data y_0 and $z_0 = y_0 + \delta_0$ and show that the perturbed solution $\delta_n = z_n - y_n$, $n \geq 0$, of Euler's method satisfies

$$\delta_n = \delta_{n-1} + ha(t_{n-1})\delta_{n-1}.$$

Thus, we can identify $\lambda = a(t_{n-1})$ and apply the absolute stability condition (2.1.16) locally. Similarly, show that the perturbed Euler solution of the nonlinear ODE (2.1.1) satisfies

$$\delta_n = \delta_{n-1} + h[f(t_{n-1}, z_{n-1}) - f(t_{n-1}, y_{n-1})].$$

If f is a smooth function of y, show that

$$\delta_n = \delta_{n-1} + h[\frac{\partial f}{\partial y}(t_{n-1}, y_{n-1})\delta_{n-1} + O(\delta_{n-1}^2)].$$

In this case, we can identify $\lambda = f_y(t_{n-1}, y_{n-1})$ and, once again, determine the region of absolute stability locally using (2.1.16). These heuristic arguments should be established by rigorous means at some stage.

3. We've already observed that error estimates computed according to *a priori* bounds such as (2.1.13) are too conservative to be used for practical step size control. Let us consider an alternate method of estimating the global errors for Euler's method that gives more precise information.

3.1. Assume that y''' exists and show that the local error of Euler's method satisfies

$$d_n = \frac{h^2}{2}y''(t_{n-1}) + \frac{h^3}{6}y'''(\xi_n), \qquad \xi_n \in (t_{n-1}, t_n).$$

3.2. Show that the global error

$$e_n = y(t_n) - y_n.$$

satisfies

$$e_n = e_{n-1} + h[e_{n-1}f_y(t_{n-1}, y(t_{n-1})) + \frac{h}{2}y''(t_{n-1}) + O(h^2)]$$

([1], pp. 13-14).

3.3. Show that the above difference equation is the Euler solution of the IVP

$$\frac{d\hat{e}}{dt} = f_y(t, y(t))\hat{e} + \frac{y''(t)}{2} + O(h), \qquad \hat{e}(0) = 0,$$

where

$$\hat{e}(t_n) = \frac{e_n}{h}, \qquad t_n = nh.$$

Neglecting the $O(h)$ term

$$\frac{d\hat{e}}{dt} = f_y(t, y(t))\hat{e} + \frac{y''(t)}{2}, \qquad \hat{e}(0) = 0.$$

3.4. ([1], p. 24.) The solution of the above equation typically furnishes more precise error information than (12). Use the solution of the above equation with the a priori estimate (2.1.13) to calculate error estimates for the IVPs

$$y' = 2ty, \qquad y' = -2ty, \qquad 0 < t < 1, \qquad y(0) = 1.$$

4. ([1], p. 24.) The solution of the IVP

$$x' = -y, \qquad x(0) = 1, \qquad y' = x, \qquad y(0) = 0,$$

is the unit circle

$$x(t) = \cos t, \qquad y(t) = \sin t.$$

4.1. Show that Euler's method

$$x_n = x_{n-1} - hy_{n-1}, \qquad y_n = y_{n-1} + hx_{n-1},$$

does not form a closed curve, but, in fact, forms a spiral when $h = 2\pi/N$.

4.2. Show that the solution of

$$x_n = x_{n-1} - hy_{n-1}, \qquad y_n = y_{n-1} + hx_n,$$

does form a closed curve and, hence, appears to provide a better approximation. (In each case, you may answer the question analytically or computationally.)

2.2 The implicit Euler method: Stiff Systems

Consider the IVP

$$y' = -\alpha(y - t^2) + 2t, \qquad t > 0, \qquad y(0) = y_0,$$

which has the solution

$$y(t) = y_0 e^{-\alpha t} + t^2.$$

Let us suppose that α is a large positive real number. In this case, the solution varies rapidly until the transient exponential term dies out. It then settles onto the slowly varying polynomial part of the solution as shown in Figure 2.2.1. The rapidly varying part of the solution is called the "inner solution." It is non-trivial in a narrow "initial layer" near $t = 0$. The slowly varying part of the solution dominates for most of the time and is called the "reduced" or "outer" solution.

Were we to solve this problem by Euler's method, we would introduce spurious oscillations unless the absolute stability condition (2.1.16) were satisfied. This would require $h\alpha \leq 2$ or $h \leq 2/\alpha$. When α is large this can be quite restrictive.

Small perturbations to the outer solution quickly decay. Since perturbations arise naturally in numerical computation, Euler's method will require small step sizes to satisfy stability condition (2.1.16) even when the solution is varying slowly (Figure 2.2.2).

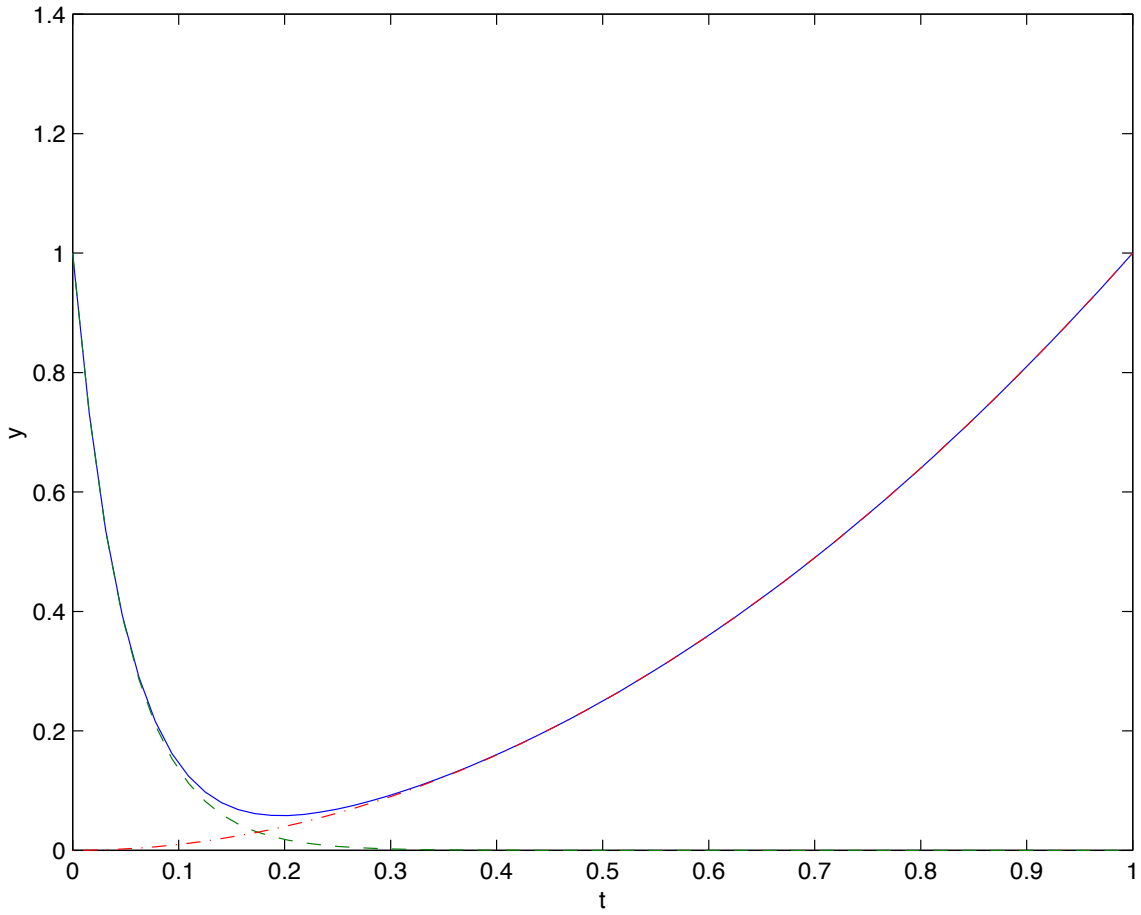

Figure 2.2.1: Solution of $y' = -\alpha(y - t^2) + 2t$ (solid) illustrating inner (dashed) and outer (dash-dot) solutions.

As shown, a small perturbation to the outer solution is introduced at $t = 0.4$. As noted, the exact solution to this perturbed solution quickly decays to the outer solution. The solution by Euler's method follows the steep initial slope of the perturbed solution and overshoots the outer solution. These overshoots and undershoots will continue in subsequent time steps as illustrated by Example 2.1.4

Problems of this type are called "stiff." They arise in applications where phenomena occur on vastly different time scales. Typical examples involve chemical kinetics, optimal control, and electrical circuits. There are many mathematical definitions of stiffness and the one that we will use follows.

Definition 2.2.1. *A problem is stiff in an interval if the step size needed for absolute stability is much smaller than the step size required for accuracy.*

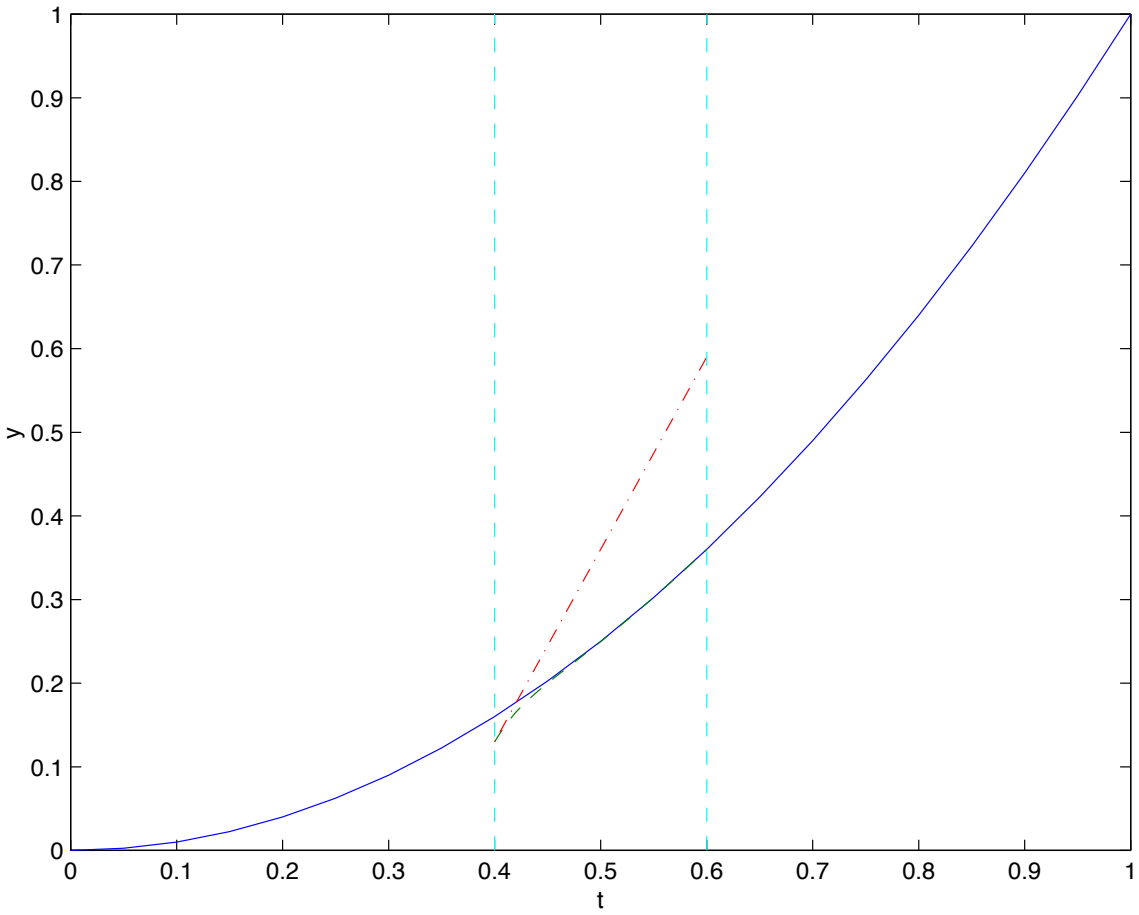

Figure 2.2.2: Euler solution (dash-dot) due to a perturbation (dashed) of $y = t^2$ (solid) introduced at time $t = 0.4$.

Stiffness not only depends on the differential equation under consideration, but the interval of interest, the accuracy criteria, and the region of absolute stability of a numerical method. For nonlinear systems

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$$

stiffness will be related to the magnitudes of the eigenvalues of the Jacobian $\mathbf{f}_{\mathbf{y}}(t, \mathbf{y})$. These eigenvalues may have vastly different sizes that vary as a function of t. Thus, detecting stiffness can be a significant problem in itself.

In order to solve stiff problems we'll need a numerical method with a weaker stability restriction than Euler's method. "Implicit methods" provide the currently accepted approach and we'll begin with the simplest implicit method, the backward (or implicit)

Euler method. Once again, consider the scalar IVP

$$y' = f(t, y), \qquad t > 0, \qquad y(0) = y_0. \tag{2.2.1}$$

As with the explicit Euler method, we'll expand the exact solution in a Taylor's series; however, this time, let us expand about the time t_n to get

$$y(t_{n-1}) = y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(\xi_n), \qquad t_{n-1} < \xi_n < t_n. \tag{2.2.2}$$

The backward Euler method is obtained by using (2.2.1) to eliminate y' and neglecting the remainder term in the Taylor series. Thus,

$$y_n = y_{n-1} + hf(t_n, y_n). \tag{2.2.3}$$

The method is called *implicit* because (2.2.3) only determines the solution when we can solve a (generally) nonlinear algebraic equation for y_n. Before discussing solutions, however, let us examine the region of absolute stability in order to determine whether or not the backward Euler method has better stability properties than the forward Euler method. Hence, let us apply (2.2.3) to the test equation (2.1.15) to obtain

$$y_n = y_{n-1} + h\lambda y_n.$$

Although implicit, (2.2.3) is simply solved for the (2.1.15) to yield

$$y_n = \frac{y_{n-1}}{1 - h\lambda}.$$

As noted in Section 2.1, since problem (2.1.15) is linear we can regard y_n as a perturbation and impose the absolute stability condition to y_n itself. In this case, y_n will not grow when

$$\frac{1}{|1 - h\lambda|} \leq 1$$

or

$$|1 - h\lambda| \geq 1. \tag{2.2.4}$$

Thus, the region of absolute stability of the backward Euler method is the region outside of a unit circle centered at $(1, 0)$ in the complex $h\lambda$ plane (Figure 2.2.3).

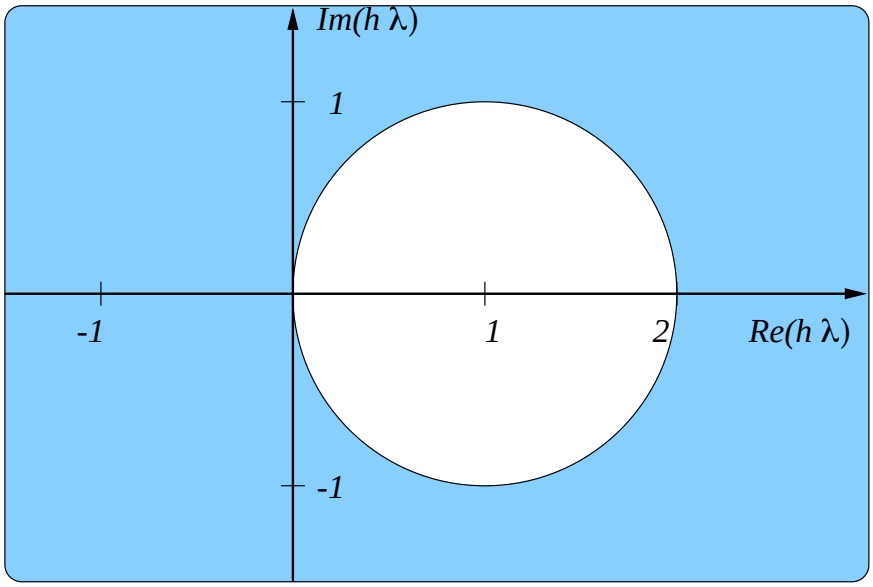

Figure 2.2.3: Region of absolute stability (shaded) of the backward Euler method.

The test equation (2.1.15) is asymptotically stable in the entire left half of the complex λ plane and stable on the imaginary axis. When approximated by the backward Euler method, it is not only absolutely stable there, but is also absolutely stable in most of the right half of the $h\lambda$ plane. Methods, such as the backward Euler method, that are stable when the differential equation is not are called *super-stable*. We'll have to see what this means.

Local errors of the backward Euler method are only slightly more difficult to estimate than for the explicit Euler method. Using (2.2.2) and (2.2.3), for example, we obtain the local discretization error as

$$\tau_n = \frac{y(t_n) - y(t_{n-1})}{h} - f(t_n, y(t_n)) = -\frac{h}{2}y''(\xi_n), \qquad t_{n-1} < \xi_n < t_n. \tag{2.2.5}$$

The local error satisfies

$$d_n = y(t_n) - y_n = y(t_n) - y(t_{n-1}) - hf(t_n, y_n)$$

or

$$d_n = h\tau_n + h[f(t_n, y(t_n)) - f(t_n, y_n)].$$

Using the mean value theorem

$$f(t_n, y(t_n)) - f(t_n, y_n) = f_y(t_n, \eta_n)(y(t_n) - y_n),$$

$y_n^{(0)} := y_{n-1}$;
$\nu := 0$;
repeat
  $y_n^{(\nu+1)} := y_{n-1} + hf(t_n, y_n^{(\nu)})$;
  $\nu := \nu + 1$
until converged;
$y_n := y_n^{(\nu+1)}$

Figure 2.2.4: Functional iteration to obtain y_n from y_{n-1} using the backward Euler method.

where η_n is between y_n and $y(t_n)$. Thus,

$$d_n = h\tau_n + hf_y(t_n, \eta_n)d_n$$

or, using (2.2.5),

$$d_n = -\frac{(h^2/2)y''(\xi_n)}{1 - hf_y(t_n, \eta_n)}. \tag{2.2.6}$$

The relationship between the local error d_n and the local discretization error τ_n is not as simple as for explicit difference equations. However, an examination of (2.1.4) reveals that the local errors of the explicit and implicit Euler methods are both $O(h^2)$.

Since y_n is defined implicitly by the backward Euler method (2.2.3), we must generally determine it by solving a nonlinear algebraic equation. This will typically require iteration. Perhaps, the simplest procedure is functional iteration as described by the algorithm of Figure 2.2.4.

We will subsequently show that functional iteration converges when

$$|hf_y(t, y)| < 1$$

for all y that occur during the iteration. Bounding $|f_y|$ by a Lipschitz constant L yields

$$|hL| \leq 1.$$

Unfortunately, L is large for stiff systems, so the step size h would have to be restricted to be small in order to achieve convergence of the iteration. This defeats the purpose of using the backward Euler method.

Newton's method provides an alternative to functional iteration that usually produces better results. Thus, let

$$F(y_n) = y_n - y_{n-1} - hf(t_n, y_n).$$

and calculate successive iterates by

$$y_n^{(\nu+1)} = y_n^{(\nu)} - \frac{F(y_n^{(\nu)})}{F_y(y_n^{(\nu)})}, \qquad \nu = 0, 1, \ldots,$$

or

$$y_n^{(\nu+1)} = y_n^{(\nu)} - \frac{y_n^{(\nu)} - y_{n-1} - hf(t_n, y_n^{(\nu)})}{1 - hf_y(t_n, y_n^{(\nu)})}, \qquad \nu = 0, 1, \ldots,$$

with

$$y_n^{(0)} = y_n.$$

Remark 1. In practice, f_y need not be reevaluated after each step of the iteration, and may approximated by finite differences as

$$f_y(t_n, y_n^{(\nu)}) \approx \frac{f(t_n, y_n^{(\nu)}) - f(t_n, y_n^{(\nu-1)})}{y_n^{(\nu)} - y_n^{(\nu-1)}}.$$

Remark 2. Newton iteration converges provided that the initial iterate $y_n^{(0)}$ is sufficiently close to y_n and h is sufficiently small. However, typically h can be much larger than that required for functional iteration.

Remark 3. It is usually not wise to perform too many Newton iterations per time step. Slow convergence of Newton's method often indicates that the time step is too large; thus, it is better to halt the iteration, reduce the time step, and calculate a new solution.

The issues raised above will be discussed in more quantitative terms in subsequent chapters. For the moment, let us reconsider the linear problem of Example 2.1.4 that we failed to solve by Euler's method because of the stability restriction (2.1.16)

Example 2.2.1. Consider the solution of the test equation

$$y' = \lambda y, \qquad y(0) = 1,$$

with $\lambda = -100$ and $h = 0.05$. Using the backward Euler method (2.2.3), we have

$$y_n = y_{n-1} + h\lambda y_n$$

n	t_n	y_n	e_n
0	0.00	1.00000	0.00000
1	0.05	0.16667	-0.15993
2	0.10	0.02778	-0.02773
3	0.15	0.00463	-0.00463
4	0.20	0.00077	-0.00077
5	0.25	0.00013	-0.00013
6	0.30	0.00002	-0.00002

Table 2.2.1: The solution and error of $y' = \lambda y$, $y(0) = 1$, obtained by the backward Euler method with $\lambda = -100$ and $h = 0.05$.

or

$$y_n = \frac{y_{n-1}}{1 - h\lambda} = \frac{y_{n-1}}{6}, \qquad n = 1, 2, \ldots, \qquad y_0 = 1.$$

Solutions are shown for $t \in [0, 0.3]$ in Table 2.2.1. Although solutions aren't particularly accurate, they do not display any of the violent oscillations that were observed with the forward Euler method.

Problems

1. When $Re(h\lambda) \ll 0$, the solution of the backward Euler equation is a much closer approximation to the solution of (2.1.15) than the forward Euler method. For simplicity, suppose λ is real and negative. Then, the ratio of solutions of the ODE at two successive times satisfies

   $$\frac{y(t_n)}{y(t_{n-1})} = e^{\lambda(t_n - t_{n-1})} = e^{h\lambda}.$$

   For $h\lambda \ll 0$, $e^{h\lambda} \ll 1$. Use (2.2.3) and a Taylor's series expansion to show that a similar ratio of successive solutions of the backward Euler method satisfies

   $$\frac{y_n}{y_{n-1}} = \frac{e^{h\lambda}}{1 - \frac{(h\lambda)^2}{2} e^{h\lambda(1-\xi/h)}}, \qquad 0 < \xi < h,$$

   which is a very good approximation of the exact ratio of solutions at successive time steps when $h\lambda \ll 0$.

   In a similar manner, the solution ratio for the explicit Euler method (2.1.2) is

   $$\frac{y_n}{y_{n-1}} = 1 + h\lambda$$

   which is clearly a poor approximation of $e^{\lambda h}$ when $h\lambda \ll 0$.

2. The local error estimate (2.2.6) can be used to establish global convergence of the backward Euler method. In particular, follow the reasoning of Theorem 2.1.1 to establish following result.

   Theorem 2.2.1. *Suppose $f(t, y)$ is continuously differentiable on*

   $$\{(t, y) \mid 0 \leq t \leq T,\ -\infty < y < \infty\}.$$

   Then the backward Euler method (2.2.3) converges to the solution of (2.2.1) at a rate of $O(h)$.

3. Consider the trapezoidal rule

   $$y_n = y_{n-1} + \frac{h}{2}[f(t_n, y_n) + f(t_{n-1}, y_{n-1})] \tag{2.2.7}$$

   for the solution of the IVP (2.2.1).

   3.1. Find expressions for its local discretization error and the local error.

   3.2. Determine its region of absolute stability.

   3.3. In order to simplify the iterative solution of (2.2.7), consider the "predictor-corrector" version of the trapezoidal rule

   $$\hat{y}_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) \tag{2.2.8a}$$

   $$y_n = y_{n-1} + \frac{h}{2}[f(t_n, \hat{y}_n) + f(t_{n-1}, y_{n-1})]. \tag{2.2.8b}$$

   Thus, the forward Euler method (2.2.8a) is used to "predict" a solution $\hat{y}_n$ at t_n and the trapezoidal rule (2.2.7b) is used to correct it. The corrector (2.2.8b) could be iteratively to "correct" the solution again, but let's suppose that this is not done.

   3.4. Determine the region of absolute stability of the predictor-corrector pair (2.2.8a,b) and compare it with the regions of absolute stability of the forward Euler method (2.1.2) and the trapezoidal rule (2.2.8). (It may be convenient to eliminate the predicted solution $\hat{y}_n$ from (2.2.8b) using (2.2.8a).)

Bibliography

[1] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice Hall, Englewood Cliffs, 1971.

[2] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, Berlin, second edition, 1993.

Chapter 3

One-Step Methods

3.1 Introduction

The explicit and implicit Euler methods for solving the scalar IVP

$$y' = f(t, y), \qquad y(0) = y_0. \tag{3.1.1}$$

have an $O(h)$ global error which is too low to make them of much practical value. With such a low order of accuracy, they will be susceptible to round off error accumulation. Additionally, the region of absolute stability of the explicit Euler method is too small. Thus, we seek higher-order methods which should provide greater accuracy than either the explicit or implicit Euler methods for the same step size. Unfortunately, there is generally a trade off between accuracy and stability and we will typically obtain one at the expense of the other.

Since we were successful in using Taylor's series to derive a method, let us proceed along the same lines, this time retaining terms of $O(h^k)$.

$$y(t_n) = y(t_{n-1}) + hy'(t_{n-1}) + \frac{h^2}{2}y''(t_{n-1}) + \ldots + \frac{h^k}{k!}y^{(k)}(t_{n-1}) + O(h^{k+1}). \tag{3.1.2}$$

Clearly, methods of this type will be explicit. Using (3.1.1)

$$y'(t_{n-1}) = f(t_{n-1}, y(t_{n-1})). \tag{3.1.3a}$$

Differentiating (3.1.1)

$$y''(t_{n-1}) = [f_t + f_y y']_{(t_{n-1}, y(t_{n-1}))} = [f_t + f_y f]_{(t_{n-1}, y(t_{n-1}))}. \tag{3.1.3b}$$

Continuing in the same manner

$$y'''(t_{n-1}) = [f_{tt} + 2f_{ty}f + f_t f_y + f_{yy}f^2 + f_y^2 f]_{(t_{n-1}, y(t_{n-1}))}, \tag{3.1.3c}$$

etc.

Specific methods are obtained by truncating the Taylor's series at different values of k. For example, if $k = 2$ we get the method

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}) + \frac{h^2}{2}[f_t + f_y f]_{(t_{n-1}, y_{n-1})}. \tag{3.1.4a}$$

From the Taylor's series expansion (3.1.2), the local error of this method is

$$d_n = \frac{h^3}{6} y'''(\xi_n). \tag{3.1.4b}$$

Thus, we succeeded in raising the order of the method. Unfortunately, methods of this type are of little practical value because the partial derivatives are difficult to evaluate for realistic problems. Any software would also have to be problem dependent. By way of suggesting an alternative, consider the special case of (3.1.1) when f is only a function of t, i.e.,

$$y' = f(t), \qquad y(0) = y_0.$$

This problem, which is of little interest, can be solved by quadrature to yield

$$y(t) = y_0 + \int_0^t f(\tau) d\tau.$$

We can easily construct high-order approximate methods for this problem by using numerical integration. Thus, for example, the simple left-rectangular rule would lead to Euler's method. The midpoint rule with a step size of h would give us

$$y(h) = y_0 + hf(h/2) + O(h^3).$$

Thus, by shifting the evaluation point to the center of the interval we obtained a higher-order approximation. Neglecting the local error term and generalizing the method to the interval $t_{n-1} < t \leq t_n$ yields

$$y_n = y_{n-1} + hf(t_{n-1} + h/2).$$

Runge [21] sought to extend this idea to true differential equations having the form of (3.1.1). Thus, we might consider

$$y_n = y_{n-1} + hf(t_n - h/2, y_{n-1/2})$$

as an extension of the simple midpoint rule to (3.1.1). The question of how to define the numerical solution $y_{n-1/2}$ at the center of the interval remains unanswered. A simple possibility that immediately comes to mind is to evaluate it by Euler's method. This gives

$$y_{n-1/2} = y_{n-1} + \frac{h}{2} f(t_{n-1}, y_{n-1});$$

however, we must verify that this approximation provides an improved order of accuracy. After all, Euler's method has an $O(h^2)$ local error and not an $O(h^3)$ error. Let's try to verify that the combined scheme does indeed have an $O(h^3)$ local error by considering the slightly more general scheme

$$y_n = y_{n-1} + h(b_1 k_1 + b_2 k_2), \tag{3.1.5a}$$

where

$$k_1 = f(t_{n-1}, y_{n-1}), \tag{3.1.5b}$$

$$k_2 = f(t_{n-1} + ch, y_{n-1} + hak_1). \tag{3.1.5c}$$

Schemes of this form are an example of *Runge-Kutta* methods. We see that the proposed midpoint scheme is recovered by selecting $b_1 = 0$, $b_2 = 1$, $c = 1/2$, and $a = 1/2$. We also see that the method does not require any partial derivatives of $f(t, y)$. Instead, (the potential) high-order accuracy is obtained by evaluating $f(t, y)$ at an additional time.

The coefficients a, b_1, b_2, and c will be determined so that a Taylor's series expansion of (3.1.5) using the exact ODE solution matches the Taylor's series expansion (3.1.2, 3.1.3) of the exact ODE solution to as high a power in h as possible. To this end, recall the formula for the Taylor's series of a function of two variables

$$F(\bar{t} + \tau, \bar{y} + \eta) = F(\bar{t}, \bar{y}) + [\tau F_t + \eta F_y]_{(\bar{t},\bar{y})} + \frac{1}{2}[\tau^2 F_{tt} + 2\tau\eta F_{ty} + \eta^2 F_{yy}]_{(\bar{t},\bar{y})} + \ldots \tag{3.1.6}$$

The expansion of (3.1.5) requires substitution of the exact solution $y(t)$ into the formula and the use of (3.1.6) to construct an expansion about $(t_{n-1}, y(t_{n-1}))$. The only term that requires any effort is k_2, which, upon insertion of the exact ODE solution, has the form

$$k_2 = f(t_{n-1} + ch, y(t_{n-1}) + haf(t_{n-1}, y(t_{n-1}))).$$

To construct an expansion, we use (3.1.6) with $F(t, y) = f(t, y)$, $\bar{t} = t_{n-1}$, $\bar{y} = y(t_{n-1})$, $\tau = ch$, and $\eta = haf(t_{n-1}, y(t_{n-1}))$. This yields

$$k_2 = f + chf_t + haff_y + \frac{1}{2}[(ch)^2 f_{tt} + 2ach^2 ff_{ty} + (ha)^2 f^2 f_{yy}] + O(h^3).$$

All arguments of f and its derivatives are $(t_{n-1}, y(t_{n-1}))$. We have suppressed these to simplify writing the expression.

Substituting the above expansion into (3.1.5a) while using (3.1.5b) with the exact ODE solution replacing y_{n-1} yields

$$y(t_n) = y(t_{n-1}) + h[b_1 f + b_2(f + chf_t + haff_y + O(h^2))]. \tag{3.1.7}$$

Similarly, substituting (3.1.3) into (3.1.2), the Taylor's series expansion of the exact solution is

$$y(t_n) = y(t_{n-1}) + hf + \frac{h^2}{2}(f_t + ff_y) + O(h^3). \tag{3.1.8}$$

All that remains is a comparison of terms of the two expansions (3.1.7) and (3.1.8). The constant terms agree. The $O(h)$ terms will agree provided that

$$b_1 + b_2 = 1. \tag{3.1.9a}$$

The $O(h^2)$ terms of the two expansions will match if

$$cb_2 = ab_2 = 1/2. \tag{3.1.9b}$$

A simple analysis would reveal that higher order terms in (3.1.7) and (3.1.8) cannot be matched. Thus, we have three equations (3.1.9) to determine the four parameters a, b_1, b_2, and c. Hence, there is a one parameter family of methods and we'll examine two specific choices.

1. Select $b_2 = 1$, then $a = c = 1/2$ and $b_1 = 0$. Using (3.1.5), this Runge-Kutta formula is

$$y_n = y_{n-1} + hk_2 \tag{3.1.10a}$$

with

$$k_1 = f(t_{n-1}, y_{n-1}), \qquad k_2 = f(t_{n-1} + h/2, y_{n-1} + hk_1/2). \tag{3.1.10b}$$

Eliminating k_1 and k_2, we can write (3.1.10) as

$$y_n = y_{n-1} + hf(t_{n-1} + h/2, y_{n-1} + hf(t_{n-1}, y_{n-1})/2), \tag{3.1.11a}$$

or

$$\hat{y}_{n-1/2} = y_{n-1} + \frac{h}{2} f(t_{n-1}, y_{n-1}), \tag{3.1.11b}$$

$$y_n = y_{n-1} + hf(t_{n-1} + h/2, \hat{y}_{n-1/2}). \tag{3.1.11c}$$

This is the midpoint rule integration formula that we discussed earlier. The $\hat{}$ on $y_{n-1/2}$ indicates that it is an intermediate rather than a final solution. As shown in Figure 3.1.1, we can regard the two-stage process (3.1.11b,c) as the result of two explicit Euler steps. The intermediate solution $\hat{y}_{n-1/2}$ is computed at $t_n + h/2$ in the first (predictor) step and this value is used to generate an approximate slope $f(t_n + h/2, \hat{y}_{n-1/2})$ for use in the second (corrector) Euler step. According to Gear [15], this method has been called the Euler-Cauchy, improved polygon, Heun, or modified Euler method. Since there seems to be some disagreement about its name and because of its similarity to midpoint rule integration, we'll call it the midpoint rule predictor-corrector.

2. Select $b_2 = 1/2$, then $a = c = 1$ and $b_1 = 1/2$. According to (3.1.5), this Runge-Kutta formula is

$$y_n = y_{n-1} + \frac{h}{2}(k_1 + k_2) \tag{3.1.12a}$$

with

$$k_1 = f(t_{n-1}, y_{n-1}), \qquad k_2 = f(t_{n-1} + h, y_{n-1} + hk_1). \tag{3.1.12b}$$

Again, eliminating k_1 and k_2,

$$y_n = y_{n-1} + \frac{h}{2}[f(t_{n-1}, y_{n-1}) + f(t_n, y_{n-1} + hf(t_{n-1}, y_{n-1}))]. \tag{3.1.13a}$$

This too, can be written as a two-stage formula

$$\hat{y}_n = y_{n-1} + hf(t_{n-1}, y_{n-1}), \tag{3.1.13b}$$

$$y_n = y_{n-1} + \frac{h}{2}[f(t_{n-1}, y_{n-1}) + f(t_n, \hat{y}_n)]. \tag{3.1.13c}$$

The formula (3.1.13a) is reminiscent of trapezoidal rule integration. The combined formula (3.1.13b,c) can, once again, be interpreted as a predictor-corrector method. Thus, as shown in Figure 3.1.2, the explicit Euler method is used to predict a solution at t_n and the trapezoidal rule is used to correct it there. We'll call (3.1.12, 3.1.13) the trapezoidal rule predictor-corrector; however, it is also known as the improved tangent, improved polygon, modified Euler, or Euler-Cauchy method ([15], Chapter 2).

Using Definition 2.1.3, we see that the Taylor's series method (3.1.4) and the Runge-Kutta methods (3.1.11) and (3.1.13) are consistent to order two since their local errors are all $O(h^3)$ (hence, their local discretization errors are $O(h^2)$).

Problems

1. Solve the IVP

$$y' = f(t, y) = y - t^2 + 1, \qquad y(0) = 1/2$$

on $0 < t \leq 1$ using the explicit Euler method and the midpoint rule. Use several step sizes and compare the error at $t = 1$ as a function of the number of evaluations of $f(t, y)$. The midpoint rule has twice the number of function evaluations of the Euler method but is higher order. Which method is preferred?

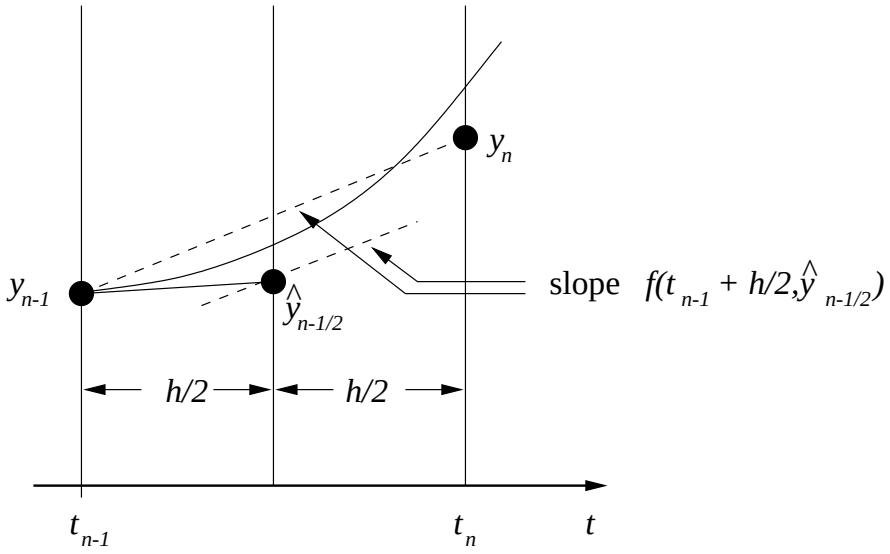

Figure 3.1.1: Midpoint rule predictor-corrector (3.1.11b,c) for one time step.

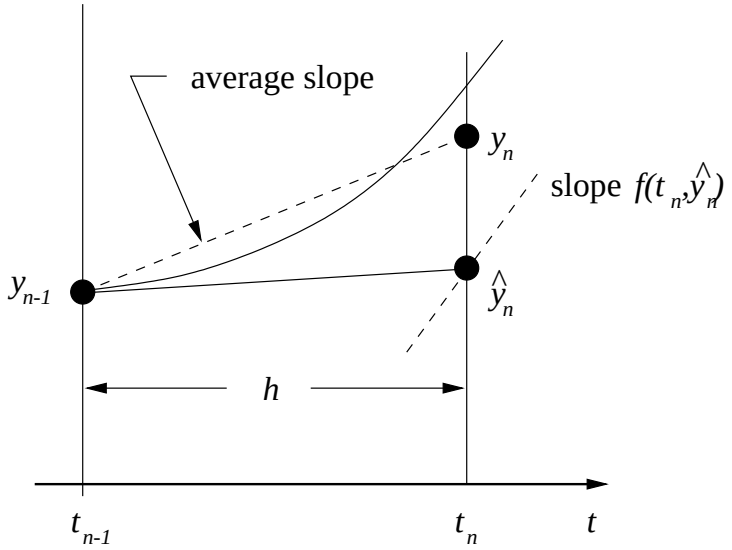

Figure 3.1.2: Trapezoidal rule predictor-corrector (3.1.13b,c) for one time step.

3.2 Explicit Runge-Kutta Methods

We would like to generalize the second order Runge-Kutta formulas considered in Section 3.1 to higher order. As usual, we will apply them to the scalar IVP (3.1.1). Runge-Kutta methods belong to a class called *one-step* methods that only require information about the solution at time t_{n-1} to calculate it at t_n. This being the case, it's possible to write

them in the general form

$$y_n = y_{n-1} + h\phi(t_{n-1}, y_{n-1}, h). \tag{3.2.1}$$

This representation is too abstract and we'll typically consider an s-stage Runge-Kutta formula for the numerical solution of the IVP (3.1.1) in the form

$$y_n = y_{n-1} + h\sum_{i=1}^{s} b_i k_i, \tag{3.2.2a}$$

where

$$k_i = f(t_{n-1} + c_i h, y_{n-1} + h\sum_{j=1}^{s} a_{ij} k_j), \qquad i = 1, 2, \ldots, s. \tag{3.2.2b}$$

These formulas are conveniently expressed as a tableau or a "Butcher diagram"

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

or more compactly as

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array}$$

We can also write (3.2.2) in the form

$$y_n = y_{n-1} + h\sum_{i=1}^{s} b_i f(t_{n-1} + c_i h, Y_i), \tag{3.2.3a}$$

where

$$Y_i = y_{n-1} + h\sum_{j=1}^{s} a_{ij} f(t_{n-1} + c_j h, Y_j), \qquad i = 1, 2, \ldots, s. \tag{3.2.3b}$$

In this form, Y_i, $i = 1, 2, \ldots, s$, are approximations of the solution at $t = t_n + c_i h$ that typically do not have as high an order of accuracy as the final solution y_n.

An explicit Runge-Kutta formula results when $a_{ij} = 0$ for $j \geq i$. Historically, all Runge-Kutta formulas were explicit; however, implicit formula are very useful for stiff systems and problems where solutions oscillate rapidly. We'll study explicit methods in this section and take up implicit methods in the next.

Runge-Kutta formulas are derived in the same manner as the second-order methods of Section 3.1. Thus, we

1. expand the exact solution of the ODE in a Taylor's series about, e.g., t_{n-1};

2. substitute the exact solution of the ODE into the Runge-Kutta formula and expanded the result in a Taylor's series about, e.g., t_{n-1}; and

3. match the two Taylor's series expansions to as high an order as possible. The coefficients are usually not uniquely determined by this process; thus, there are families of methods having a given order.

A Runge-Kutta method that is consistent to order k (or simply of order k) will match the terms of order h^k in both series. Clearly the algebra involved in obtaining these formulas increases combinatorically with increasing order. A symbolic manipulation system, such as MAPLE or MATHEMATICA, can be used to reduce complexity. Fortunately, the derivation is adequately demonstrated by the second-order methods presented in Section 3.1 and, for the most part, we will not need to present detailed derivations of higher-order methods.

There are three one-parameter families of three-stage, third-order explicit Runge-Kutta methods [3, 16]. However, the most popular explicit methods are of order four. Their tableau has the general form

0	0	0	0	0
c_2	a_{21}	0	0	0
c_3	a_{31}	a_{32}	0	0
c_4	a_{41}	a_{42}	a_{43}	0
	b_1	b_2	b_3	b_4

The Taylor's series produce eleven equations for the thirteen nonzero parameters listed above. The *classical* Runge-Kutta method has the following form:

$$y_n = y_{n-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{3.2.4a}$$

where

$$k_1 = f(t_{n-1}, y_{n-1}), \tag{3.2.4b}$$

$$k_2 = f(t_{n-1} + h/2, y_{n-1} + hk_1/2), \tag{3.2.4c}$$

Order, k	1	2	3	4	5	6	7	8	9
Min. Fn. Evals.	1	2	3	4	6	7	9	11	11

Table 3.2.1: Minimum number of function evaluations for explicit Runge-Kutta methods of various orders.

$$k_3 = f(t_{n-1} + h/2, y_{n-1} + hk_2/2), \tag{3.2.4d}$$

$$k_4 = f(t_{n-1} + h, y_{n-1} + hk_3). \tag{3.2.4e}$$

Some observations about this method follow:

1. The local error of (3.2.4) is $O(h^5)$. In order to get an *a priori* estimate of the local error we have to subtract the two Taylor's series representations of the solution. This is very tedious and typically does not yield a useful result. Runge-Kutta methods do not yield simple *a priori* error estimates.

2. Four function evaluations are required per time step.

3. In the (unlikely) case when f is a function of t only, (3.2.4) reduces to

$$y_n = y_{n-1} + \frac{h}{6}[f(t_{n-1}) + 4f(t_{n-1/2}) + f(t_n)],$$

which is the same as Simpson's rule integration.

Our limited experience with Runge-Kutta methods would suggest that the number of function evaluations increases linearly with the order of the method. Unfortunately, Butcher [8] showed that this is not the case. Some key results are summarized in Table 3.2.1. The popularity of the four-stage, fourth-order Runge-Kutta methods are now clear. From Table 3.2.1, we see that a fifth order Runge-Kutta method requires an additional two function evaluations per step. Additionally, Butcher [8] showed that an explicit s-stage Runge-Kutta method will have an order of at least $s - 2$.

Although Runge-Kutta formulas are tedious to derive, we can make a few general observations. An order one formula must be exact when the solution of the ODE is a linear polynomial. Were this not true, it wouldn't annihilate the constant and linear

terms in a Taylor's series expansion of the exact ODE solution and, hence, could not have the requisite $O(h^2)$ local error to be first-order accurate. Thus, the Runge-Kutta method should produce exact solutions of the differential equations $y' = 0$ and $y' = 1$. The constant-solution condition is satisfied identically by construction of the Runge-Kutta formulas. Using (3.2.3a), the latter (linear-solution) condition with $y(t) = t$ and $f(t, y) = 1$ implies

$$t_n = t_{n-1} + h \sum_{i=1}^{s} b_i$$

or

$$\sum_{i=1}^{s} b_i = 1. \tag{3.2.5a}$$

If we also require the intermediate solutions Y_i to be first order, then the use of (3.2.3b) with $Y_i = t_{n-1} + c_i h$ gives

$$c_i = \sum_{j=1}^{s} a_{ij}, \qquad i = 1, 2, \ldots, s. \tag{3.2.5b}$$

This condition does not have to be satisfied for low-order Runge-Kutta methods [16]; however, its satisfaction simplifies the task of obtaining order conditions for higher-order methods. Methods that satisfy (3.2.5b) also treat autonomous and non-autonomous systems in a symmetric manner (Problem 1).

We can continue this process to higher orders. Thus, the Runge-Kutta method will be of order p if it is exact when the differential equation and solution are

$$y' = (t - t_{n-1})^{l-1}, \qquad y(t) = \frac{1}{l}(t - t_{n-1})^l, \qquad l = 1, 2, \ldots, p.$$

(The use of $t - t_{n-1}$ as a variable simplifies the algebraic manipulations.) Substituting these solutions into (3.2.3a) implies that

$$\frac{h^l}{l} = h \sum_{j=1}^{s} b_i (c_i h)^{l-1}$$

or

$$\sum_{i=1}^{s} b_i c_i^{l-1} = \frac{1}{l}, \qquad l = 1, 2, \ldots, p. \tag{3.2.5c}$$

Conditions (3.2.5c) are necessary for a method to be order p, but may not be sufficient. Note that there is no dependence on the coefficients a_{ij} $i, j = 1, 2, \ldots, s$, in formulas (3.2.5a,c). This is because our strategy of examining simple differential equations is not matching all possible terms in a Taylor's series expansion of the solution. This, as noted, is a tedious operation. Butcher developed a method of simplifying the work by constructing rooted trees that present the order conditions in a graphical way. They are discussed in many texts (e.g., [11, 16]); however, they are still complex and we will not pursue them here. Instead, we'll develop additional necessary order conditions by considering the simple ODE

$$y' = y.$$

Replacing $f(t, y)$ in (3.2.3) by y yields

$$y_n = y_{n-1} + h \sum_{i=1}^{s} b_i Y_i,$$

$$Y_i = y_{n-1} + h \sum_{j=1}^{s} a_{ij} Y_j, \qquad i = 1, 2, \ldots, s.$$

It's simpler to use vector notation

$$y_n = y_{n-1} + h\mathbf{b}^T\mathbf{Y}, \qquad \mathbf{Y} = y_{n-1}\mathbf{l} + h\mathbf{A}\mathbf{Y}$$

where

$$\mathbf{Y} = [Y_1,\ Y_2,\ \ldots,\ Y_s]^T, \tag{3.2.6a}$$

$$\mathbf{l} = [1,\ 1,\ \ldots,\ 1]^T, \tag{3.2.6b}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1s} \\ a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \ldots & a_{ss} \end{bmatrix}, \tag{3.2.6c}$$

and

$$\mathbf{b} = [b_1,\ b_2,\ \ldots,\ b_s]^T. \tag{3.2.6d}$$

Eliminating $\mathbf{Y}$, we have

$$\mathbf{Y} = y_{n-1}(\mathbf{I} - h\mathbf{A})^{-1}\mathbf{l}$$

and

$$y_n = y_{n-1} + hy_{n-1}\mathbf{b}^T(\mathbf{I} - h\mathbf{A})^{-1}\mathbf{l}.$$

Assuming that y_{n-1} is exact, the exact solution of this test equation is $y_n = e^h y_{n-1}$. Expanding this solution and $(\mathbf{I} - h\mathbf{A})^{-1}$ in series

$$1 + h + \ldots + \frac{h^k}{k!} + \ldots = 1 + h\mathbf{b}^T(\mathbf{I} + h\mathbf{A} + \ldots + h^k\mathbf{A}^k + \ldots)\mathbf{l}.$$

Equating like powers of h yields the order condition

$$\mathbf{b}^T\mathbf{A}^{k-1}\mathbf{l} = \frac{1}{k!}, \qquad k = 1, 2, \ldots, p. \tag{3.2.7}$$

We recognize that this condition with $k = 1$ is identical to (3.2.5a). Letting $\mathbf{c} = [c_1,\ c_2,\ \ldots,\ c_s]^T$, we may write (3.2.5c) with $l = 2$ in the form $\mathbf{b}^T\mathbf{c} = 1/2$. The vector form of (3.2.5b) is $\mathbf{Al} = \mathbf{c}$. Thus, $\mathbf{b}^T\mathbf{Al} = 1/2$, which is the same as (3.2.7) with $k = 2$. Beyond $k = 2$, the order conditions (3.2.5c) and (3.2.7) are independent.

Although conditions (3.2.5) and (3.2.7) are only necessary for a method to be of order p, they are sufficient in many cases. The actual number of conditions for a Runge-Kutta method of order p are presented in Table 3.2.2 [16]. These results assume that (3.2.5b) has been satisfied.

Order, p	1	2	3	4	5	6	7	8	9	10
No. of Conds.	1	2	4	8	17	37	85	200	486	1205

Table 3.2.2: The number of conditions for a Runge-Kutta method of order p [16].

Theorem 3.2.1. *The necessary and sufficient conditions for a Runge-Kutta method (3.2.3) to be of second order are (3.2.5c), $l = 1, 2$, and (3.2.7), $k = 2$. If (3.2.5b) is satisfied then (3.2.5), $k = 1, 2$, are necessary and sufficient for second-order accuracy.*

Proof. We require numerous Taylor's series expansions. To begin, we expand $f(t_{n-1} + c_i h, Y_i)$ using (3.1.6) to obtain

$$f(t_{n-1} + c_i h, Y_i) = f + f_t c_i h + f_y(Y_i - y(t_{n-1})) + \frac{1}{2}[f_{tt}(c_i h)^2 + 2f_{ty}(c_i h)(Y_i - y(t_{n-1}))+$$

$$f_{yy}(Y_i - y(t_{n-1}))^2] + O(h^3).$$

All arguments of f and its derivatives are at $(t_{n-1}, y(t_{n-1}))$. They have been suppressed for simplicity.

Substituting the exact ODE solution and the above expression into (3.2.3a) yields

$$y(t_n) = y(t_{n-1}) + h\sum_{i=1}^{s} b_i[f + f_t c_i h + f_y(Y_i - y(t_{n-1})) + O(h^2)].$$

The expansion of $Y_i - y(t_{n-1})$ will, fortunately, only require the leading term; thus, using (3.2.3b)

$$Y_i - y(t_{n-1}) = h\sum_{j=1}^{s} a_{ij} f + O(h^2).$$

Hence, we have

$$y(t_n) = y(t_{n-1}) + h\sum_{i=1}^{s} b_i[f + f_t c_i h + hf f_y \sum_{j=1}^{s} a_{ij} + O(h^2)].$$

Equating terms of this series with the Taylor's series (3.1.8) of the exact solution yields (3.2.5c) with $l = 1$, (3.2.5c) with $l = 2$, and (3.2.7) with $k = 2$. We have demonstrated the equivalence of these conditions when (3.2.5b) is satisfied. □

Remark 1. The results of Theorem 3.2.1 and conditions (3.2.5) and (3.2.7) apply to both explicit and implicit methods.

Let us conclude this section with a brief discussion of the absolute stability of explicit methods. We will present a more detailed analysis in Section 3.4; however, the present material will serve to motivate the need for implicit methods. Thus, consider an s-stage explicit Runge-Kutta method applied to the test equation

$$y' = \lambda y. \tag{3.2.8}$$

Using (3.2.8) in (3.2.3) with the simplification that $a_{ij} = 0$, $j \geq i$, for explicit methods yields

$$y_n = y_{n-1} + z\sum_{i=1}^{s} b_i Y_i = y_{n-1} + z\mathbf{b}^T\mathbf{Y} \tag{3.2.9a}$$

where

$$Y_i = y_{n-1} + z\sum_{j=1}^{i-1} a_{ij}Y_j, \qquad i = 1, 2, \ldots, s, \tag{3.2.9b}$$

and

$$z = h\lambda. \tag{3.2.9c}$$

The vector form of (3.2.9) is

$$\mathbf{Y} = y_{n-1}\mathbf{l} + z\mathbf{A}\mathbf{Y}. \tag{3.2.9d}$$

Using this to eliminate $\mathbf{Y}$ in (3.2.9a), we have

$$y_n = y_{n-1}[1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{l}].$$

Expanding the inverse

$$y_n = y_{n-1}[1 + z\mathbf{b}^T(\mathbf{I} + z\mathbf{A} + \ldots + z^k\mathbf{A}^k + \ldots)\mathbf{l}]$$

Using (3.2.7)

$$y_n = R(z)y_{n-1} \tag{3.2.10a}$$

where

$$R(z) = 1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!} + \sum_{j=p+1}^{\infty} z^j\mathbf{b}^T\mathbf{A}^{j-1}\mathbf{l}.$$

The matrix $\mathbf{A}$ is strictly lower triangular for an s-stage explicit Runge-Kutta method; thus, $\mathbf{A}^{j-1} = \mathbf{0}$, $j > s$. Therefore,

$$R(z) = 1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!} + \sum_{j=p+1}^{s} z^j\mathbf{b}^T\mathbf{A}^{j-1}\mathbf{l}. \tag{3.2.10b}$$

In particular, for explicit s-stage methods with $p = s \leq 4$, we have

$$R(z) = 1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!}, \qquad s = p \leq 4. \tag{3.2.10c}$$

The exact solution of the test equation (3.2.8) is

$$y(t_n) = e^{h\lambda}y(t_{n-1});$$

thus, as expected, a *pth*-order Runga-Kutta formula approximates a Taylor's series expansion of the exact solution through terms of order p.

Using Definition 2.1.7 and (3.2.10), the region of absolute stability of an explicit Runge-Kutta method is

$$|R(z)| = \left|1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!} + \sum_{j=p+1}^{s} z^j \mathbf{b}^T \mathbf{A}^{j-1} \mathbf{l}\right| \leq 1. \tag{3.2.11a}$$

In particular,

$$|R(z)| = \left|1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!}\right| \leq 1, \qquad s = p \leq 4. \tag{3.2.11b}$$

Since no Runge-Kutta coefficients appear in (3.2.11b), we have the interesting result.

Lemma 3.2.1. *All p-stage explicit Runge-Kutta methods of order $p \leq 4$ have the same region of absolute stability.*

Since $|e^{i\theta}| = 1$, $0 \leq \theta \leq 2\pi$, we can determine the boundary of the absolute stability regions (3.2.11a,b) by solving the nonlinear equation

$$R(z) = e^{i\theta}. \tag{3.2.12}$$

Clearly, (3.2.12) implies that $|y_n/y_{n-1}| = 1$. For $p = 1$ (i.e., for Euler's method), the boundary of the absolute-stability region is determined as

$$1 + z = e^{i\theta},$$

which can easily be recognized as the familiar unit circle centered at $z = -1 + 0i$. For real values of z the intervals of absolute stability for methods with $p = s \leq 4$ are shown in Table 3.2.3. Absolute stability regions for complex values of z are illustrated for the same methods in Figure 3.2.1. Methods are stable within the closed regions shown. The regions of absolute stability grow with increasing p. When $p = 3, 4$, they also extend slightly into the right half of the complex z-plane.

Problems

1. Instead of solving the IVP (3.1.1), many software systems treat an autonomous ODE $y' = f(y)$. Non-autonomous ODEs can be written as autonomous systems

Order, p	Interval of Absolute Stability
1	(-2,0)
2	(-2,0)
3	(-2.51,0)
4	(-2.78,0)

Table 3.2.3: Interval of absolute stability for p-stage explicit Runge-Kutta methods of order $p = 1, 2, 3, 4$.

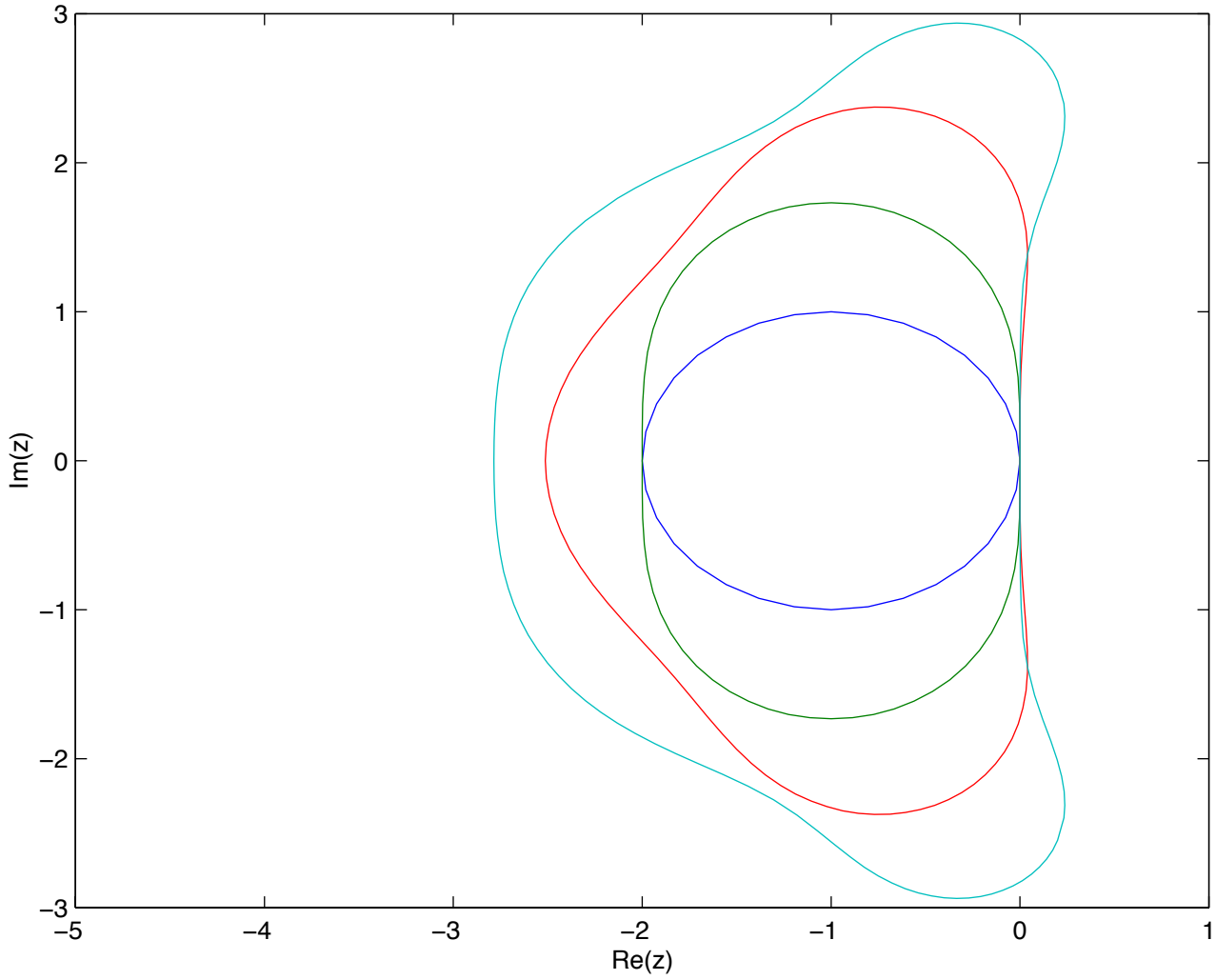

Figure 3.2.1: Region of absolute stability for p-stage explicit Runge-Kutta methods of order $p = 1, 2, 3, 4$ (interiors of smaller closed curves to larger ones).

by letting t be a dependent variable satisfying the ODE $t' = 1$. A Runge-Kutta method for an autonomous ODE can be obtained from, e.g., (3.2.3) by dropping the time terms, i.e.,

$$y_n = y_{n-1} + h \sum_{i=1}^{s} b_i f(Y_i),$$

with

$$Y_i = y_{n-1} + h \sum_{j=1}^{s} a_{ij} f(Y_j), \qquad i = 1, 2, \ldots, s.$$

The Runge-Kutta evaluation points c_i, $i = 1, 2, \ldots, s$, do not appear in this form. Show that Runge-Kutta formulas (3.2.3) and the one above will handle autonomous and non-autonomous systems in the same manner when (3.2.5b) is satisfied.

3.3 Implicit Runge-Kutta Methods

We'll begin this section with a negative result that will motivate the need for implicit methods.

Lemma 3.3.1. *No explicit Runge-Kutta method can have an unbounded region of absolute stability.*

Proof. Using (3.2.10), the region of absolute stability of an explicit Runge-Kutta method satisfies

$$|y_n / y_{n-1}| = |R(z)| \leq 1, \qquad z = h\lambda,$$

where $R(z)$ is a polynomial of degree s, the number of stages of the method. Since $R(z)$ is a polynomial, $|R(z)| \to \infty$ as $|z| \to \infty$ and, thus, the stability region is bounded. □

Hence, once again, we turn to implicit methods as a means of enlarging the region of absolute stability.

Necessary order conditions for s-stage implicit Runge-Kutta methods are given by (3.2.5c, 3.2.7) (with sufficient conditions given in Hairer *et al.* [16], Section II.2). A condition on the maximum possible order follows.

Theorem 3.3.1. *The maximum order of an implicit s-stage Runge-Kutta method is $2s$.*

Proof. cf. Butcher [7]. □

The derivations of implicit Runge-Kutta methods follow those for explicit methods. We'll derive the simplest method and then give a few more examples.

Example 3.3.1. Consider the implicit 1-stage method obtained from (3.2.3) with $s = 1$ as

$$y_n = y_{n-1} + hb_1 f(t_{n-1} + c_1 h, Y_1), \tag{3.3.1a}$$

$$Y_1 = y_{n-1} + ha_{11} f(t_{n-1} + c_1 h, Y_1). \tag{3.3.1b}$$

To determine the coefficients c_1, b_1, and a_{11}, we substitute the exact ODE solution into (3.3.1a,b) and expand (3.3.1a) in a Taylor's series

$$y(t_n) = y(t_{n-1}) + hb_1[f + c_1 h f_t + f_y(Y_1 - y(t_{n-1})) + O(h^2)],$$

where $f := f(t_{n-1}, y(t_{n-1}))$, etc. Expanding (3.3.1b) in a Taylor's series and substituting the result into the above expression yields

$$y(t_n) = y(t_{n-1}) + hb_1[f + c_1 h f_t + ha_{11} f f_y + O(h^2)].$$

Comparing the terms of the above series with the Taylor's series

$$y(t_n) = y(t_{n-1}) + hf + \frac{h^2}{2}(f_t + f f_y) + O(h^3)$$

of the exact solution yields

$$b_1 = 1, \qquad a_{11} = c_1 = \frac{1}{2}.$$

Substituting these coefficients into (3.3.1), we find the method to be an implicit midpoint rule

$$y_n = y_{n-1} + hf(t_{n-1} + h/2, Y_1), \tag{3.3.2a}$$

$$Y_1 = y_{n-1} + \frac{h}{2} f(t_{n-1} + h/2, Y_1). \tag{3.3.2b}$$

The tableau for this method is

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

The formula has similarities to the midpoint rule predictor-corrector (3.1.11); however, there are important differences. Here, the backward Euler method (rather than the forward Euler method) may be regarded as furnishing a predictor (3.3.2b) with the midpoint rule providing the corrector (3.3.2a). However, formulas (3.3.2a) and (3.3.2b) are coupled and must be solved simultaneously rather than sequentially.

Example 3.3.2. The two-stage method having maximal order four presented in the following tableau was developed by Hammer and Hollingsworth [18].

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

This method is derived in Gear [15], Section 2.5.

Example 3.3.3. Let us examine the region of absolute stability of the implicit midpoint rule (3.3.2). Thus, applying (3.3.2) to the test equation (3.2.8) we find

$$Y_1 = y_{n-1} + \frac{h\lambda}{2} Y_1$$

and

$$y_n = y_{n-1} + h\lambda Y_1.$$

Solving for Y_1

$$Y_1 = \frac{y_{n-1}}{1 - h\lambda/2},$$

and eliminating it in order to explicitly determine y_n as

$$y_n = \left(1 + \frac{h\lambda}{1 - h\lambda/2}\right) y_{n-1} = \left(\frac{1 + h\lambda/2}{1 - h\lambda/2}\right) y_{n-1}.$$

Thus, the region of absolute stability is interior to the curve

$$\frac{1 + z/2}{1 - z/2} = e^{i\theta}, \qquad z = h\lambda.$$

Solving for z

$$z = 2\frac{1 - e^{i\theta}}{1 + e^{i\theta}} = -2\frac{e^{i\theta/2} - e^{-i\theta/2}}{e^{i\theta/2} + e^{-i\theta/2}} = -2i \tan \theta/2.$$

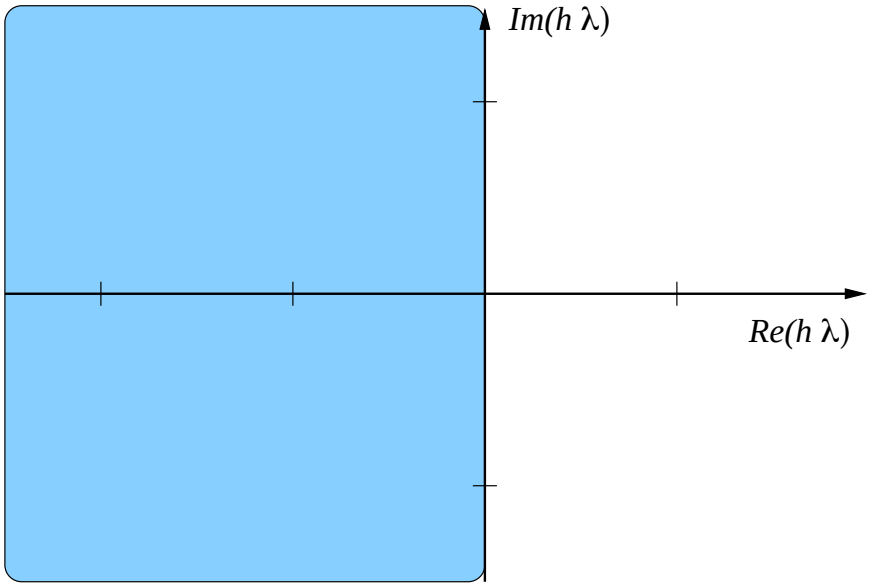

Figure 3.3.1: Region of absolute stability for the implicit midpoint rule (3.3.2).

Since z is imaginary, the implicit midpoint rule is absolutely stable in the entire negative half of the complex z plane (Figure 3.3.1).

Let us generalize the absolute stability analysis presented in Example 3.3.3 before considering additional methods. This analysis will be helpful since we will be interested in developing methods with very large regions of absolute stability. Thus, we apply the general method (3.2.3) to the test equation (3.2.8) to obtain

$$y_n = y_{n-1} + z\mathbf{b}^T\mathbf{Y}, \tag{3.3.3a}$$

$$(\mathbf{I} - z\mathbf{A})\mathbf{Y} = y_{n-1}\mathbf{l}, \tag{3.3.3b}$$

where $\mathbf{Y}$, $\mathbf{l}$, $\mathbf{A}$, and $\mathbf{b}$ and are defined by (3.2.6) and $z = h\lambda$.

Eliminating $\mathbf{Y}$ in (3.3.3a) by using (3.3.3b) we find

$$y_n = R(z)y_{n-1}, \tag{3.3.4a}$$

where

$$R(z) = 1 + z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{l}. \tag{3.3.4b}$$

The region of absolute stability is the set of all complex z where $|R(z)| \leq 1$. While $R(z)$ is a polynomial for an explicit method, it is a rational function for an implicit method.

Hence, the region of absolute stability can be unbounded. As shown in Section 3.2, a method of order p will satisfy

$$R(z) = e^z + O(z^{p+1}).$$

Rational-function approximations of the exponential are called Padé approximations.

Definition 3.3.1. *The (j,k) Padé approximation $R_{jk}(z)$ is the maximum-order approximation of e^z having the form*

$$R_{jk}(z) = \frac{P_k(z)}{Q_j(z)} = \frac{p_0 + p_1 z + \ldots + p_k z^k}{q_0 + q_1 z + \ldots + q_j z^j}, \tag{3.3.5a}$$

where P_k and Q_j have no common factors,

$$Q_j(0) = q_0 = 1, \tag{3.3.5b}$$

and

$$R_{jk}(z) = e^z + O(z^{k+j+1}). \tag{3.3.5c}$$

With R_{jk} normalized by (3.3.5b), there are $k + j + 1$ undetermined parameters in (3.3.5a) that can be determined by matching the first $k + j + 1$ terms in the Taylor's series expansion of e^z. Thus, the error of the approximation should be $O(z^{k+j+1})$. Using (3.3.5c), we have

$$\sum_{i=0}^{k+j} \frac{z^i}{i!} = \frac{\sum_{i=0}^{k} p_i z^i}{\sum_{i=0}^{j} q_i z^i} + O(z^{k+j+1}) \tag{3.3.6}$$

Equating the coefficients of like powers of z determines the parameters p_i, $i = 0, 1, \ldots, k$ and q_i, $i = 1, 2, \ldots, j$.

Example 3.3.4. Find the (2,0) Padé approximation of e^z. Setting $j = 2$ and $k = 0$ in (3.3.6) gives

$$(1 + z + \frac{z^2}{2})(1 + q_1 z + q_2 z^2) = p_0.$$

Equating the coefficients of z^i, $i = 0, 1, 2$, gives

$$p_0 = 1, \qquad 1 + q_1 = 0, \qquad \frac{1}{2} + q_1 + q_2 = 0;$$

Thus,

$$p_0 = 1, \qquad q_1 = -1, \qquad q_2 = 1/2.$$

Using (3.3.5), the (2,0) Padé approximation is

$$R_{20}(z) = \frac{1}{1 - z + z^2/2}.$$

Additionally,

$$e^z = R_{20}(z) + O(z^3).$$

Some other Padé approximations are presented in Table 3.3.1. We recognize that the (0,1) approximation corresponds to Euler's method, the (1,0) method corresponds to the backward Euler method, and the (1,1) approximation corresponds to the midpoint rule. (The (1,1) approximation also corresponds to the trapezoidal rule.) Methods corresponding to the (s, s) diagonal Padé approximations are Butcher's maximum order implicit Runge-Kutta methods (Theorem 3.3.1).

	$k = 0$	1	2
$j = 0$	1	$1 + z$	$1 + z + z^2/2$
1	$\frac{1}{1-z}$	$\frac{1+z/2}{1-z/2}$	$\frac{1+2z/3+z^2/6}{1-z/3}$
2	$\frac{1}{1-z+z^2/2}$	$\frac{1+z/3}{1-2z/3+z^2/6}$	$\frac{1+z/2+z^2/12}{1-z/2+z^2/12}$

Table 3.3.1: Some Padé approximations of e^z.

Theorem 3.3.2. *There is one and only one $2s$-order s-stage implicit Runge-Kutta formula and it corresponds to the (s, s) Padé approximation.*

Proof. cf. Butcher [7]. □

We'll be able to construct several implicit Runge-Kutta methods having unbounded absolute-stability regions. We'll want to characterize these methods according to their behavior as $|z| \to \infty$ and this requires some additional notions of stability.

Definition 3.3.2. *A numerical method is A-stable if its region of absolute stability includes the entire left-half plane $Re(h\lambda) \leq 0$.*

The relationship between A-stability and the Padé approximations is established by the following theorem.

Theorem 3.3.3. *Methods that lead to a diagonal or one of the first two sub-diagonals of the Padé table for e^z are A-stable.*

Proof. The proof appears in Ehle [13]. Without introducing additional properties of Padé approximations, we'll make some observations using the results of Table 3.3.1.

1. We have shown that the regions of absolute stability of the backward Euler method and the midpoint rule include the entire left-half of the $h\lambda$ plane; hence, they are A-stable.

2. The coefficients of the highest-order terms of $P_s(z)$ and $Q_s(z)$ are the same for diagonal Padé approximations $R_{ss}(z)$; hence, $|R_{ss}(z)| \to 1$ as $|z| \to \infty$ and these methods are A-stable (Table 3.3.1).

3. For the sub-diagonal (1,0) and (2,1) Padé approximations, $|R(z)| \to 0$ as $|z| \to \infty$ and these methods will also be A-stable.

□

It is quite difficult to find high-order A-stable methods. Implicit Runge-Kutta methods provide the most viable approach. Examining Table 3.3.1, we see that we can introduce another stability notion.

Definition 3.3.3. *A numerical method is L-stable if it is A-stable and if $|R(z)| \to 0$ as $|z| \to \infty$.*

The backward Euler method and, more generally, methods corresponding to sub-diagonal Padé approximations in the first two bands are L-stable ([17], Section IV.4). L-stable methods are preferred for stiff problems where $Re(\lambda) \ll 0$ but methods where $|R(z)| \to 1$ are more suitable when $Re(\lambda) \approx 0$ but $|Im(\lambda)| \gg 1$, i.e., when solutions oscillate rapidly.

Explicit Runge-Kutta methods are easily solved, but implicit methods will require an iterative solution. Since implicit methods will generally be used for stiff systems,

Newton's method will be preferred to functional iteration. To emphasize the difficulty, we'll illustrate Runge-Kutta methods of the form (3.2.3) for vector IVPs

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(0) = \mathbf{y}_0, \tag{3.3.7}$$

where $\mathbf{y}$, *etc.* are m-vectors. The application of (3.2.3) to vector systems just requires the use of vector arithmetic; thus,

$$\mathbf{Y}_i = \mathbf{y}_{n-1} + h \sum_{j=1}^{s} a_{ij} \mathbf{f}(t_{n-1} + c_j h, \mathbf{Y}_j), \qquad i = 1, 2, \ldots, s, \tag{3.3.8a}$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h \sum_{i=1}^{s} b_i \mathbf{f}(t_{n-1} + c_i h, \mathbf{Y}_i). \tag{3.3.8b}$$

Once again, $\mathbf{y}_n$ *etc.* are m-vectors.

To use Newton's method, we write the nonlinear system (3.3.8a) in the form

$$\mathbf{F}_i(\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_s) = \mathbf{Y}_i - \mathbf{y}_{n-1} - h \sum_{j=1}^{s} a_{ij} \mathbf{f}(t_{n-1} + h c_j, \mathbf{Y}_j) = 0, \qquad j = 1, 2, \ldots, s, \tag{3.3.9a}$$

and get

$$\begin{bmatrix} \mathbf{I} - a_{11}\mathbf{J}_1^{(\nu)} & -ha_{12}\mathbf{J}_2^{(\nu)} & & -ha_{1s}\mathbf{J}_s^{(\nu)} \\ -ha_{21}\mathbf{J}_1^{(\nu)} & \mathbf{I} - ha_{22}\mathbf{J}_2^{(\nu)} & & -ha_{2s}\mathbf{J}_s^{(\nu)} \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{s1}\mathbf{J}_1^{(\nu)} & -ha_{s2}\mathbf{J}_2^{(\nu)} & & \mathbf{I} - ha_{ss}\mathbf{J}_s^{(\nu)} \end{bmatrix} \begin{bmatrix} \delta\mathbf{Y}_1^{(\nu)} \\ \delta\mathbf{Y}_2^{(\nu)} \\ \vdots \\ \delta\mathbf{Y}_s^{(\nu)} \end{bmatrix} = - \begin{bmatrix} \mathbf{F}_1^{(\nu)} \\ \mathbf{F}_2^{(\nu)} \\ \vdots \\ \mathbf{F}_s^{(\nu)} \end{bmatrix}, \tag{3.3.9b}$$

$$\mathbf{Y}_i^{(\nu+1)} = \mathbf{Y}_i^{(\nu)} + \delta\mathbf{Y}_i^{(\nu)}, \qquad i = 1, 2, \ldots, s, \qquad \nu = 0, 1, \ldots, \tag{3.3.9c}$$

where

$$\mathbf{J}_j^{(\nu)} = \mathbf{f}_{\mathbf{y}}(t_{n-1} + hc_j, \mathbf{Y}_j^{(\nu)}), \qquad \mathbf{F}_j^{(\nu)} = \mathbf{F}_j(\mathbf{Y}_1^{(\nu)}, \mathbf{Y}_2^{(\nu)}, \ldots, \mathbf{Y}_s^{(\nu)}), \qquad j = 1, 2, \ldots, s. \tag{3.3.9d}$$

For an s-stage Runge-Kutta method applied to an m-dimensional system (3.3.7), the Jacobian in (3.3.9b) has dimension $sm \times sm$. This will be expensive for high-order methods and high-dimensional ODEs and will only be competitive with, e.g., implicit

multistep methods (Chapter 5) under special conditions. Some simplifications are possible and these can reduce the work. For example, we can approximate all of the Jacobians as

$$\mathbf{J} = \mathbf{f}_{\mathbf{y}}(t_{n-1}, \mathbf{y}_{n-1}). \tag{3.3.10a}$$

In this case, we can even shorten the notation by introducing the *Kronecker* or *direct product* of two matrices as

$$\mathbf{A} \otimes \mathbf{J} = \begin{bmatrix} a_{11}\mathbf{J} & a_{12}\mathbf{J} & & a_{1s}\mathbf{J} \\ a_{21}\mathbf{J} & a_{22}\mathbf{J} & & a_{2s}\mathbf{J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1}\mathbf{J} & a_{s2}\mathbf{J} & & a_{ss}\mathbf{J} \end{bmatrix}. \tag{3.3.10b}$$

Then, (3.3.9b) can be written concisely as

$$(\mathbf{I} - h\mathbf{A} \otimes \mathbf{J})\delta\mathbf{Y}^{(\nu)} = -\mathbf{F}^{(\nu)} \tag{3.3.10c}$$

where $\mathbf{A}$ was given by (3.2.6c) and

$$\delta\mathbf{Y}^{(\nu)} = \begin{bmatrix} \delta\mathbf{Y}_1^{(\nu)} \\ \delta\mathbf{Y}_2^{(\nu)} \\ \vdots \\ \delta\mathbf{Y}_s^{(\nu)} \end{bmatrix}, \qquad \mathbf{F}^{(\nu)} = \begin{bmatrix} \mathbf{F}_1^{(\nu)} \\ \mathbf{F}_2^{(\nu)} \\ \vdots \\ \mathbf{F}_s^{(\nu)} \end{bmatrix}. \tag{3.3.10d}$$

The approximation of the Jacobian does not change the accuracy of the computed solution, only the convergence rate of the iteration. As long as convergence remains good, the same Jacobian can be used for several time step and only be re-evaluated when convergence of the Newton iteration slows.

Even with this simplification, with m ranging into the thousands, the solution of (3.3.10) is clearly expensive and other ways of reducing the computational cost are necessary. *Diagonally implicit Runge-Kutta (DIRK) methods* offer one possibility. A DIRK method is one where $a_{ij} = 0$, $i < j$ and at least one $a_{ii} \neq 0$, $i, j = 1, 2, \ldots, s$. If, in addition, $a_{11} = a_{22} = \ldots = a_{ss} = a$, the technique is known as a *singly diagonally implicit Runge-Kutta (SDIRK)* method. Thus, the coefficient matrix of an SDIRK method has

the form

$$\mathbf{A} = \begin{bmatrix} a & & & \\ a_{21} & a & & \\ \vdots & \vdots & \ddots & \\ a_{s1} & a_{s2} & & a \end{bmatrix}. \tag{3.3.11}$$

Thus, with the approximation (3.3.10), the system Jacobian in (3.3.10c) is

$$(\mathbf{I} - h\mathbf{A} \otimes \mathbf{J}) = \begin{bmatrix} \mathbf{I} - ha\mathbf{J} & & & 0 \\ -ha_{21}\mathbf{J} & \mathbf{I} - ha\mathbf{J} & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{s1}\mathbf{J} & -ha_{s2}\mathbf{J} & & \mathbf{I} - ha\mathbf{J} \end{bmatrix}.$$

The Newton system (3.3.10) is lower block triangular and can be solved by forward substitution. Thus, the first block of (3.3.10c) is solved for $\delta\mathbf{Y}_1^{(\nu)}$. Knowing Y_1 the second equation is solved for $\delta\mathbf{Y}_2^{(\nu)}$, etc. The Jacobian J is the same for all stages; thus, the diagonal blocks need only be factored once by Gaussian elimination and forward and backward substitution may be used for each solution.

The implicit midpoint rule (3.3.2) is a one-stage, second-order DIRK method. We'll examine a two-stage DIRK method momentarily, but first we note that the maximum order of an s-stage DIRK method is $s + 1$ [2].

Example 3.3.5. A two-stage DIRK formula has the tableau

$$\begin{array}{c|cc} c_1 & a_{11} & 0 \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array}$$

and it could be of third order. According to Theorem 3.2.1, the conditions for second-order accuracy are (3.2.5c) with $l = 1, 2$ when (3.2.5b) is satisfied, i.e.,

$$b_1 + b_2 = 1, \qquad c_1 = a_{11}, \qquad c_2 = a_{21} + a_{22}, \qquad b_1c_1 + b_2c_2 = 1/2.$$

(As noted earlier, satisfaction of (3.2.5b) is not necessary, but it simplifies the algebraic manipulations.) We might guess that the remaining conditions necessary for third order accuracy are (3.2.5c) with $l = 3$ and (3.2.7) with $k = 3$, i.e.,

$$b_1c_1^2 + b_2c_2^2 = 1/3$$

and

$$\mathbf{b}^T\mathbf{A}^2\mathbf{l} = \mathbf{bAc} = b_1a_{11}c_1 + b_2(a_{21}c_1 + a_{22}c_2) = 1/6,$$

where (3.2.5b) was used to simplify the last expression. After some effort, this system of six equations in seven unknowns can be solved to yield

$$c_2 = \frac{2-3c_1}{3-6c_1}, \qquad b_1 = \frac{c_2 - 1/2}{c_2 - c_1}, \qquad b_2 = \frac{1/2 - c_1}{c_2 - c_1},$$

$$a_{11} = c_1, \qquad a_{22} = \frac{1/6 - b_1 c_1^2}{b_2(c_2 - c_1)}, \qquad a_{21} = c_2 - a_{22}.$$

As written, the solution is parameterized by c_1. Choosing $c_1 = 1/3$ gives

$$\begin{array}{c|cc} 1/3 & 1/3 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 3/4 & 1/4 \end{array}$$

Using (3.2.3), the method is

$$Y_1 = y_{n-1} + \frac{h}{3} f(t_{n-1} + \frac{h}{3}, Y_1), \qquad Y_2 = y_{n-1} + \frac{h}{2}[f(t_{n-1} + \frac{h}{3}, Y_1) + f(t_n, Y_2)],$$

$$y_n = y_{n-1} + \frac{h}{4}[3f(t_{n-1} + \frac{h}{3}, Y_1) + f(t_n, Y_2)].$$

We can check by constructing a Taylor's series that this method is indeed third order. Hairer *et al.* [16], Section II.2, additionally show that our necessary conditions for third-order accuracy are also sufficient in this case.

The computation of Y_1 can be recognized as the backward Euler method for one-third of the time step h. The computation of Y_2 and y_n are not recognizable in terms of simple quadrature rules. Since the method is third-order, its local error is $O(h^4)$.

We can also construct an SDIRK method by insisting that $a_{11} = a_{22}$. Enforcing this condition and using the previous relations gives two methods having the tableau

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1-\gamma & 1-2\gamma & \gamma \\ \hline & 1/2 & 1/2 \end{array}$$

where

$$\gamma = \frac{1}{2}(1 \pm \frac{1}{\sqrt{3}}).$$

The method with $\gamma = (1 + 1/\sqrt{3})/2$ is A-stable while the other method has a bounded stability region. Thus, this would be the method of choice.

Let us conclude this Section by noting a relationship between implicit Runge-Kutta and collocation methods. With $u(t)$ a polynomial of degree s in t for $t \geq t_{n-1}$, a *collocation method* for the IVP

$$y' = f(t, y), \qquad y(t_{n-1}) = y_{n-1} \tag{3.3.12a}$$

consists of solving

$$u(t_{n-1}) = y_{n-1} \tag{3.3.12b}$$

$$u'(t_{n-1} + c_i h) = f(t_{n-1} + c_i h, u(t_{n-1} + c_i h)), \qquad i = 1, 2, \ldots, s, \tag{3.3.12c}$$

where c_i, $i = 1, 2, \ldots, s$, are non-negative parameters. Thus, the collocation method consists of satisfying the ODE exactly at s points. The solution $u(t_{n-1} + h)$ may be used as the initial condition y_n for the next time step.

Usually, the collocation points $t_{n-1} + c_i h$ are such that $c_i \in [0, 1]$, $i = 1, 2, \ldots, s$, but this need not be the case [6, 10, 20].

Generally, the c_i, $i = 1, 2, \ldots, s$, are distinct and we shall assume that this is the case here. (The coefficients need not be distinct when the approximation $u(t)$ interpolates some solution derivatives, e.g., as with Hermite interpolation.) Approximating $u'(t)$, $t \geq t_{n-1}$, by a Lagrange interpolating polynomial of degree $s - 1$, we have

$$u'(t) = \sum_{j=1}^{s} k_j L_j(\frac{t - t_{n-1}}{h}) \tag{3.3.13a}$$

where

$$L_j(\tau) = \prod_{i=1, i \neq j}^{s} \frac{\tau - c_i}{c_j - c_i}, \tag{3.3.13b}$$

$$\tau = \frac{t - t_{n-1}}{h}. \tag{3.3.13c}$$

The polynomials $L_j(\tau)$, $j = 1, 2, \ldots, s$, are a product of $s - 1$ linear factors and are, hence, of degree $s - 1$. They satisfy

$$L_j(c_i) = \delta_{ji}, \qquad j, i = 1, 2, \ldots, s, \tag{3.3.13d}$$

where δ_{ji} is the Kronecker delta. Using (3.3.13a), we see that $u'(t)$ satisfies the interpolation conditions

$$u'(t_{n-1} + c_i h) = k_i, \qquad i = 1, 2, \ldots, s. \tag{3.3.13e}$$

Transforming variables in (3.3.13a) using (3.3.13c)

$$u(t_{n-1} + \tau h) = y_{n-1} + h \int_0^\tau u'(t_{n-1} + \sigma h) d\sigma. \tag{3.3.14}$$

By construction, (3.3.14) satisfies (3.3.12b). Substituting (3.3.13e) and (3.3.14) into (3.3.12c), we have

$$k_i = f(t_{n-1} + c_i h, y_{n-1} + h \sum_{j=1}^{s} k_j \int_0^{c_i} L_j(\sigma) d\sigma).$$

This formula is identical to the typical Runge-Kutta formula (3.2.2b) provided that

$$a_{ij} = \int_0^{c_i} L_j(\sigma) d\sigma. \tag{3.3.15a}$$

Similarly, using (3.3.13a) in (3.3.14) and evaluating the result at $\tau = 1$ yields

$$u(t_{n-1} + h) = y_n = y_{n-1} + h \sum_{j=1}^{s} k_j \int_0^1 L_j(\sigma) d\sigma.$$

This formula is identical to (3.2.2a) provided that

$$b_j = \int_0^1 L_j(\sigma) d\sigma. \tag{3.3.15b}$$

This view of a Runge-Kutta method as a collocation method is useful in many situations. Let us illustrate one result.

Theorem 3.3.4. *A Runge-Kutta method with distinct c_i, $i = 1, 2, \ldots, s$, and of order at least s is a collocation method satisfying (3.3.12), (3.3.15) if and only if it satisfies the order conditions*

$$\sum_{j=1}^{s} a_{ij} c_j^{q-1} = \frac{c_i^q}{q}, \qquad i, q = 1, 2, \ldots, s. \tag{3.3.16}$$

Remark 1. The order conditions (3.3.15) are related to the previous conditions (3.2.5c, 3.2.7) (cf. [16], Section II.7).

Proof. We use the Lagrange interpolating polynomial (3.3.13) to represent *any* polynomial $P(\tau)$ of degree $s-1$ as

$$P(\tau) = \sum_{j=1}^{s} P(c_j) L_j(\tau).$$

Regarding $P(\tau)$ as $u'(t_n + \tau h)$, integrate to obtain

$$u(t_{n-1} + c_i h) - y_{n-1} = \int_0^{c_i} P(\tau) d\tau = \sum_{j=1}^{s} P(c_j) \int_0^{c_i} L_j(\tau) d\tau, \qquad i = 1, 2, \ldots, s.$$

Assuming that (3.3.15a) is satisfied, we have

$$\int_0^{c_i} P(\tau) d\tau = \sum_{j=1}^{s} a_{ij} P(c_j), \qquad i = 1, 2, \ldots, s.$$

Now choose $P(\tau) = \tau^{q-1}$, $q = 1, 2, \ldots, s$, to obtain (3.3.16). The proof of the converse follows the same arguments (cf. [16], Section II.7). □

Now, we might ask if there is an optimal way of selecting the collocation points. Appropriate strategies would select them so that accuracy and/or stability are maximized. Let's handle accuracy first. The following theorems discuss relevant accuracy issues.

Theorem 3.3.5. *(Alekseev and Gröbner) Let x, y, and z satisfy*

$$x'(t, \sigma, z(\sigma)) = f(t, x(t, \sigma, z(\sigma))), \qquad x(\sigma, \sigma, z(\sigma)) = z(\sigma), \tag{3.3.17a}$$

$$y'(t) = f(t, y(t)), \qquad y(0) = y_0, \tag{3.3.17b}$$

$$z'(t) = f(t, z(t)) + g(t, z(t)), \qquad z(0) = y_0, \tag{3.3.17c}$$

with $f_y(t, y) \in C^0$, $t > 0$. Then,

$$z(t) - y(t) = \int_0^t \frac{\partial x(t, \sigma, z(\sigma))}{\partial z} g(\sigma, z(\sigma)) d\sigma. \tag{3.3.17d}$$

Remark 2. Formula (3.3.17d) is often called the *nonlinear variation of parameters.*

Remark 3. The parameter σ identifies the time that the initial conditions are applied in (3.3.17a). A prime, as usual, denotes t differentiation.

Remark 4. Observe that $y(t) = x(t, 0, y_0)$.

Proof. cf. Hairer *et al.* [16], Section I.14, and Problem 1. □

Theorem 3.3.5 makes it easy for us to associate the collocation error with a quadrature error as indicated below.

Theorem 3.3.6. *Consider the quadrature rule*

$$\int_{t_{n-1}}^{t_n} F(t)dt = h\int_0^1 F(t_{n-1}+\tau h)d\tau = h\sum_{i=1}^{s} b_i F(t_{n-1}+c_i h) + E_p \tag{3.3.18a}$$

where

$$E_p = Ch^{p+1}F^{(p)}(\xi_n), \qquad \xi_n \in (t_{n-1}, t_n), \tag{3.3.18b}$$

$F \in C^p(t_{n-1}, t_n)$*, and* C *is a constant. Then the collocation method (3.3.12) has order* p*.*

Proof. Consider the identity

$$u' = f(t,u) + [u' - f(t,u)]$$

and use Theorem 3.3.5 on $[t_{n-1}, t_n]$ with $z(t) = u(t)$ and $g(t,u) = u' - f(t,u)$ to obtain

$$u(t_n) - y(t_n) = \int_{t_{n-1}}^{t_n} x_u(t_n, \sigma, u(\sigma))[u'(\sigma) - f(\sigma, u(\sigma))]d\sigma.$$

Replace this integral by the quadrature rule (3.3.18) to obtain

$$u(t_n) - y(t_n) = h\sum_{i=1}^{s} b_i x_u(t_n, t_{n-1}+c_i h, u(t_{n-1}+c_i h))[u'(t_{n-1}+c_i h) -$$

$$f(t_{n-1}+c_i h, u(t_{n-1}+c_i h))] + E_p.$$

All terms in the summation vanish upon use of the collocation equations (3.3.12); thus,

$$|u(t_n) - y(t_n)| = |E_p| \le |C|h^{p+1} \max_{\sigma\in[t_{n-1},t_n]} |\frac{\partial^p}{\partial\sigma^p} x_u(t_n, \sigma, u(\sigma))[u'(\sigma) - f(\sigma, u)]|.$$

It remains to show that the derivatives in the above expression are bounded as $h \to 0$. We'll omit this detail which is proven in Hairer *et al.* [16], Section II.7. Thus,

$$|y(t_n) - u(t_n)| \le \hat{C}h^{p+1} \tag{3.3.19}$$

and the collocation method (3.3.12) is of order p. □

At last, our task is clear. We should select the collocation points c_i, $i = 1, 2, \ldots, s$, to maximize the order p of the quadrature rule (3.3.18). We'll review some of the details describing the derivation of (3.3.18). Additional material appears in most elementary numerical analysis texts [4]. Let $\hat{F}(\tau) = F(t_{n-1} + \tau h)$ and approximate it by a Lagrange interpolating polynomial of degree $s - 1$ to obtain

$$\hat{F}(\tau) = \sum_{j=1}^{s} \hat{F}(c_j) L_j(\tau) + \frac{M_s(\tau)}{s!} \hat{F}^{(s)}(\xi), \qquad \xi \in (0, 1), \tag{3.3.20a}$$

where

$$M_s(\tau) = \prod_{i=1}^{s} (\tau - c_i). \tag{3.3.20b}$$

(Differentiation in (3.3.20a) is with respect to τ, not t.)

Integrate (3.3.20a) and use (3.3.15b) to obtain

$$\int_0^1 \hat{F}(\tau) d\tau = \sum_{j=1}^{s} b_j \hat{F}(c_j) + \hat{E}_s \tag{3.3.21a}$$

where

$$\hat{E}_s = \frac{1}{s!} \int_0^1 M_s(\tau) \hat{F}^{(s)}(\xi(\tau)) d\tau = \frac{1}{s!} \int_0^1 \prod_{i=1}^{s} (\tau - c_i) \hat{F}^{(s)}(\xi(\tau)) d\tau. \tag{3.3.21b}$$

In Newton-Cotes quadrature rules, such as the trapezoidal and Simpson's rules, the evaluation points c_i, $i = 1, 2, \ldots, s$, are specified *a priori*. With Gaussian quadrature, however, the points are selected to maximize the order of the rule. This can be done by expanding $\hat{F}^{(s)}(\xi(\tau))$ in a Taylor's series and selecting the c_i, $i = 1, 2, \ldots, s$, to annihilate as many terms as possible. Alternatively, and equivalently, the quadrature rule can be designed to integrate polynomials exactly to as high a degree as possible. The actual series expansion is complicated by the fact that $\hat{F}^{(s)}$ is evaluated at $\xi(\tau)$ in (3.3.21b). Isaacson and Keller [19] provide additional details on this matter; however, we'll sidestep the subtleties by assuming that all derivatives of $\xi(\tau)$ are bounded so that $\hat{F}^{(s)}$ has an expansion in powers of τ of the form

$$\hat{F}^{(s)}(\tau) = \alpha_0 + \alpha_1 \tau + \ldots + \alpha_{r-1} \tau^{r-1} + O(\tau^r).$$

d	$P_d(x)$
0	1
1	x
2	$x^2 - \frac{1}{3}$
3	$x^3 - \frac{3x}{5}$
4	$x^4 - \frac{6x^2}{7} + \frac{3}{35}$
5	$x^5 - \frac{10x^3}{9} + \frac{5x}{21}$

Table 3.3.2: Legendre polynomials $P_d(x)$ of degree $d \in [0, 5]$ on $-1 \leq x \leq 1$.

The first r terms of this series will be annihilated by (3.3.21b) if $M_s(\tau)$ is orthogonal to polynomials of degree $r - 1$, i.e., if

$$\int_0^1 M(\tau)\tau^{q-1}d\tau = 0, \qquad q = 1, 2, \ldots, r. \tag{3.3.22}$$

Under these conditions, were we to transform the integrals in (3.3.21) and (3.3.22) back to t dependence using (3.3.13c), we would obtain the error of (3.3.18b) with $p = s + r$. With the s coefficients c_i, $i = 1, 2, \ldots, s$, we would expect the maximum value of r to be s. According to Theorem 3.3.6, this choice would lead to a collocation method of order $2s$, i.e., a method having $p = r + s = 2s$ and an $O(h^{2s+1})$ local error. These are Butcher's maximal order formulas (Theorem 3.3.1) corresponding to the diagonal Padé approximations.

The maximum-order coefficients identified above are the roots of the s *th*-degree Legendre polynomial scaled to the interval $(0, 1)$. The first six Legendre polynomials are listed in Table 3.3.2. Additional polynomials and their roots appear in Abromowitz and Stegun [1], Chapter 22.

Example 3.3.6. According to Table 3.3.2, the roots of $P_2(x)$ are $x_{1,2} = \pm 1/\sqrt{3}$ on $[-1, 1]$. Mapping these to $[0, 1]$ by the linear transformation $\tau = (1 + x)/2$, we obtain the collocation points for the maximal-order two-stage method as

$$c_1 = \frac{1}{2}(1 - \frac{1}{\sqrt{3}}), \qquad c_2 = \frac{1}{2}(1 + \frac{1}{\sqrt{3}}).$$

Since this is our first experience with these techniques, let us verify our results by a direct evaluation of (3.3.22) using (3.3.20b); thus,

$$\int_0^1 (\tau - c_1)(\tau - c_2)d\tau = 0, \qquad \int_0^1 (\tau - c_1)(\tau - c_2)\tau d\tau = 0.$$

Integrating

$$\frac{1}{3} - \frac{c_1 + c_2}{2} + c_1 c_2 = 0, \qquad \frac{1}{4} - \frac{c_1 + c_2}{3} + \frac{c_1 c_2}{2} = 0.$$

These may easily be solved to confirm the collocation points obtained by using the roots of $P_2(x)$. In this case, we recognize c_1 and c_2 as the evaluation points of the Hammer-Hollingsworth formula of Example 3.3.2.

With the collocation points c_i, $i = 1, 2, \ldots, s$, determined, the coefficients a_{ij} and b_j, $i, j = 1, 2, \ldots, s$, may be determined from (3.3.15a,b). These maximal order collocation formulas are A-stable since they correspond to diagonal Padé approximations (Theorem 3.3.3).

We may not want to impose the maximal order conditions to obtain, e.g., better stability and computational properties. With Radau quadrature, we fix one of the coefficients at an endpoint; thus, we set either $c_1 = 0$ or $c_s = 1$. The choice $c_1 = 0$ leads to methods with bounded regions of absolute stability. Thus, the methods of choice have $c_s = 1$. They correspond to the subdiagonal Padé approximations and are, hence, A- and L-stable (Theorem 3.3.3). They have orders of $p = 2s - 1$ [17], Section IV.5. Such excellent stability and accuracy properties makes these methods very popular for solving stiff systems.

The Radau polynomial of degree s on $-1 \leq x \leq 1$ is

$$R_s(x) = P_s(x) - \frac{s}{2s-1} P_{s-1}(x).$$

The roots of R_s transformed to $[0, 1]$ (using $\tau = (1 + x)/2$) are the c_i, $i + 1, 2, \ldots, s$. All values of c_i, $i = 1, 2, \ldots, s$, are on $(0, 1]$ with, as designed, $c_s = 1$. The one-stage Radau method is the backward Euler method. The tableau of the two-stage Radau method is (Problem 2)

$\frac{1}{3}$	$\frac{5}{12}$	- $\frac{1}{12}$
1	$\frac{3}{4}$	$\frac{1}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$

We'll conclude this Section with a discussion of singly implicit Runge-Kutta (SIRK) methods. These methods are of order s, which is less than the Legendre $(2s)$, Radau $(2s-1)$, and DIRK $(s+1)$ techniques. They still have excellent A- and L-stability properties and, perhaps, offer a computational advantage.

A SIRK method is one where the coefficient matrix $\mathbf{A}$ has a single s-fold real eigenvalue. These collocation methods were Originally developed by Butcher [9] and have been subsequently extended [5, 10, 6, 20]. Collocating, as described, leads to the system (3.3.12-3.3.15). The intermediate solutions Y_i, $i = 1, 2, \ldots, s$, have the vector form specified by (3.2.9d) with the elements of $\mathbf{A}$ given by (3.3.15a). Multiplying (3.2.9d) by a nonsingular matrix $\mathbf{T}^{-1}$, we obtain

$$\mathbf{T}^{-1}\mathbf{Y} = y_n\mathbf{T}^{-1}\mathbf{l} + h\mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{T}^{-1}\mathbf{f}$$

where $\mathbf{Y}$, $\mathbf{l}$, $\mathbf{A}$, and $\mathbf{f}$ are, respectively, given by (3.2.6a-c) and

$$\mathbf{f} = \begin{bmatrix} f(t_{n-1} + c_1 h) \\ f(t_{n-1} + c_2 h) \\ \vdots \\ f(t_{n-1} + c_s h) \end{bmatrix}. \tag{3.3.23}$$

Let

$$\hat{\mathbf{Y}} = \mathbf{T}^{-1}\mathbf{Y}, \qquad \hat{\mathbf{l}} = \mathbf{T}^{-1}\mathbf{l}, \qquad \hat{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}, \qquad \hat{\mathbf{f}} = \mathbf{T}^{-1}\mathbf{f}. \tag{3.3.24}$$

Butcher [9] chose the collocation points $c_i = \alpha\xi_i$, $i = 1, 2, \ldots, s$, where ξ_i is the $i\ th$ root of the $s\ th$-degree Laguerre polynomial $L_s(t)$ and α is chosen so that the numerical method has favorable stability properties. butcher also selected $\mathbf{T}$ to have elements

$$T_{ij} = L_{i-1}(\xi_j).$$

Then

$$\hat{\mathbf{A}} = \begin{bmatrix} \lambda & & & \\ \lambda & \lambda & & \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \end{bmatrix}. \tag{3.3.25}$$

Thus, $\hat{\mathbf{A}}$ is lower bidiagonal with the single eigenvalue λ. The linearized system (3.3.9) is easily solved in the transformed variables. (A similar transformation also works with Radau methods [17].) Butcher [9] and Burrage [5] show that it is possible to find A-stable SIRK methods for $s \leq 8$. These methods are also L-stable with the exception of the seven-stage method.

Problems

1. Verify that (3.3.17d) is correct when $f(t, y) = ay$ with a a constant.

2. Consider the θ method

$$y_n = y_{n-1} + h[(1-\theta)f(t_{n-1}, y_{n-1}) + \theta f(t_n, y_n)]$$

with $\theta \in [0, 1]$. The method corresponds to the Euler method with $\theta = 0$, the trapezoidal rule with $\theta = 1/2$, and the backward Euler method and when $\theta = 1$.

2.1. Write the Runge-Kutta tableau for this method.

2.2. For what values of θ is the method A-stable? Justify your answer.

3. Radau or Lobatto quadrature rules have evaluation points at one or both endpoints of the interval of integration, respectively. Consider the two two-stage Runge-Kutta methods based on collocation at Radau points. In one, the collocation point $c_1 = 0$ and in the other the collocation point $c_2 = 1$. In each case, the other collocation point (c_2 for the first method and c_1 for the second method) is to be determined so that the resulting method has as high an order of accuracy as possible.

3.1. Determine the parameters a_{ij}, b_j, and c_i, $i, j = 1, 2$ for the two collocation methods and identify their orders of accuracy.

3.2. To which elements of the Padé table do these methods correspond?

3.3. Determine the regions of absolute stability for these methods? Are the methods A- and/or L-stable?

3.4 Convergence, Stability, Error Estimation

The concepts of convergence, stability, and *a priori* error estimation introduced in Chapter 2 readily extend to a general class of (explicit or implicit) one-step methods having the form

$$y_n = y_{n-1} + h\phi(t_{n-1}, y_{n-1}, h). \tag{3.4.1a}$$

Again, consider the scalar IVP

$$y' = f(t, y), \qquad y(0) = y_0, \tag{3.4.1b}$$

and, to begin, we'll show that one-step methods are stable when ϕ satisfies a Lipschitz condition on y.

Theorem 3.4.1. *If $\phi(t, y, h)$ satisfies a Lipschitz condition on y then the one-step method (3.4.1a) is stable.*

Proof. The analysis follows the lines of Theorem 2.1.1. Let y_n and z_n satisfy method (3.4.1) and

$$z_n = z_{n-1} + h\phi(t_{n-1}, z_{n-1}, h), \qquad z_0 = y_0 + \delta_0, \tag{3.4.2}$$

respectively. Subtracting (3.4.2) from (3.4.1)

$$y_n - z_n = y_{n-1} - z_{n-1} + h[\phi(t_{n-1}, y_{n-1}, h) - \phi(t_{n-1}, z_{n-1}, h)].$$

Using the Lipschitz condition

$$|y_n - z_n| \le (1 + hL)|y_{n-1} - z_{n-1}|.$$

Iterating the above inequality leads to

$$|y_n - z_n| \le (1 + hL)^n |y_0 - z_0|.$$

Using (2.1.10)

$$|y_n - z_n| \le e^{nhL}|\delta_0| \le e^{LT}\delta \le k\delta,$$

since $nh \le T$ and $|\delta_0| \le \delta$. □

Example 3.4.1. The function ϕ satisfies a Lipschitz condition whenever f does. Consider, for example, the explicit midpoint rule which has the form of (3.4.1a) with

$$\phi(t, y, h) = f(t + h/2, y + hf(t, y)/2).$$

Then,

$$|\phi(t, y, h) - \phi(t, z, h)| = |f(t + h/2, y + hf(t, y)/2) - f(t + h/2, z + hf(t, z)/2)|$$

Using the Lipschitz condition on f

$$|\phi(t, y, h) - \phi(t, z, h)| \leq L|y + hf(t, y)/2 - z - hf(t, z)/2|$$

or

$$|\phi(t, y, h) - \phi(t, z, h)| \leq L[|y - z| + (h/2)|f(t, y) - f(t, z)|]$$

or

$$|\phi(t, y, h) - \phi(t, z, h)| \leq L(1 + hL/2)|y - z|.$$

Thus, we can take the Lipschitz constant for ϕ to be $L(1 + \hat{h}L/2)$ for $h \in (0, \hat{h}]$.

In addition to a Lipschitz condition, convergence of the one-step method (3.4.1a) requires consistency. Recall (Definition 2.1.3), that consistency implies that the local discretization error $\lim_{h\to 0} \tau_n = 0$. Consistency is particularly simple for a one-step method.

Lemma 3.4.1. *The one-step method (3.4.1a) is consistent with the ODE $y' = f(t, y)$ if*

$$\phi(t, y, 0) = f(t, y). \tag{3.4.3}$$

Proof. The local discretization error of (3.4.1a) satisfies

$$\tau_n = \frac{y(t_n) - y(t_{n-1})}{h} - \phi(t_{n-1}, y(t_{n-1}), h).$$

Letting h tend to zero

$$\lim_{h\to 0} \tau_n = y'(t_{n-1}) - \phi(t_{n-1}, y(t_{n-1}), 0).$$

Using the ODE to replace y' yields the result. □

Theorem 3.4.2. *Let $\phi(t,y,h)$ be a continuous function of t, y, and h on $0 \le t \le T$, $-\infty < y < \infty$, and $0 \le h \le \hat{h}$, respectively, and satisfy a Lipschitz condition on y. Then the one-step method (3.4.1a) converges to the solution of (3.4.1b) if and only if it is consistent.*

Proof. Let $z(t)$ satisfy the IVP

$$z' = \phi(t,z,0), \qquad z(0) = y_0, \tag{3.4.4}$$

and let z_n, $n \ge 0$, satisfy

$$z_n = z_{n-1} + h\phi(t_{n-1}, z_{n-1}, h), \qquad n \ge 0, \qquad z_0 = y_0. \tag{3.4.5}$$

Using the mean value theorem and (3.4.4)

$$z(t_n) - z(t_{n-1}) = hz'(t_{n-1} + h\theta_n) = h\phi(t_{n-1} + h\theta_n, z(t_{n-1} + h\theta_n), 0), \tag{3.4.6}$$

where $\theta_n \in (0,1)$. Let

$$e_n = z(t_n) - z_n \tag{3.4.7}$$

and subtract (3.4.5) from (3.4.6) to obtain

$$e_n = e_{n-1} + h[\phi(t_{n-1} + h\theta_n, z(t_{n-1} + h\theta_n), 0) - \phi(t_{n-1}, z_{n-1}, h)].$$

Adding and subtracting similar terms

$$\begin{aligned} e_n = e_{n-1} \quad &+ \quad h[\phi(t_{n-1} + h\theta_n, z(t_{n-1} + h\theta_n), 0) - \phi(t_{n-1}, z(t_{n-1}), 0) \\ &+ \quad \phi(t_{n-1}, z(t_{n-1}), h) - \phi(t_{n-1}, z_{n-1}, h) \\ &+ \quad \phi(t_{n-1}, z(t_{n-1}), 0) - \phi(t_{n-1}, z(t_{n-1}), h)]. \end{aligned} \tag{3.4.8a}$$

Using the Lipschitz condition

$$|\phi(t_{n-1}, z(t_{n-1}), h) - \phi(t_{n-1}, z_{n-1}, h)| \le L|e_n|. \tag{3.4.8b}$$

Since $\phi(t,y,h) \in C^0$, it is uniformly continuous on the compact set $t \in [0,T]$, $y = z(t)$, $h \in [0,\hat{h}]$; thus,

$$\sigma(h) = \max_{t\in[0,T]} |\phi(t_{n-1}, z(t_{n-1}), 0) - \phi(t_{n-1}, z(t_{n-1}), h)| = O(h). \tag{3.4.8c}$$

Similarly,

$$\tau(h) = \max_{t\in[0,T]} |\phi(t_{n-1} + h\theta_n, z(t_{n-1} + h\theta_n), 0) - \phi(t_{n-1}, z(t_{n-1}), 0)| = O(h). \qquad (3.4.8d)$$

Substituting (3.4.8b,c,d) into (3.4.8a)

$$|e_n| \le |e_{n-1}| + h[L|e_{n-1}| + \sigma(h) + \tau(h)]. \qquad (3.4.9)$$

Equation (3.4.9) is a first order difference inequality with constant (independent of n) coefficients having the general form

$$|e_n| \le A|e_{n-1}| + B \qquad (3.4.10a)$$

where, in this case,

$$A = 1 + hL, \qquad (3.4.10b)$$

$$B = h[\sigma(h) + \tau(h)]. \qquad (3.4.10c)$$

The solution of (3.4.10a) is

$$|e_n| \le A^n|e_0| + \left(\frac{A^n - 1}{A - 1}\right) B, \qquad n \ge 0.$$

Since $e_0 = 0$, we have

$$|e_n| \le \left(\frac{(1 + hL)^n - 1}{hL}\right) h[\sigma(h) + \tau(h)],$$

or, using (2.1.10)

$$|e_n| \le \left(\frac{e^{LT} - 1}{L}\right) [\sigma(h) + \tau(h)].$$

Both $\sigma(h)$ and $\tau(h)$ approach zero as $h \to 0$; therefore,

$$\lim_{h\to 0,\ n\to\infty\ Nh=T} z_n = z(t_n).$$

Thus, z_n converges to $z(t_n)$, where $z(t)$ is the solution of (3.4.4). If the one-step method satisfies the consistency condition (3.4.3), then $z(t) = y(t)$. Thus, y_n converges to $y(t_n)$, $n \ge 0$. This establishes sufficiency of the consistency condition for convergence.

In order to show that consistency is necessary for convergence, assume that the one-step method (3.4.1a) converges to the solution of the IVP (3.4.1b). Then, $y_n \to y(t_n)$ for all $t \in [0,T]$ as $h \to 0$ and $N \to \infty$. Now, z_n, defined by (3.4.5), is identical to y_n, so z_n must also converge to $y(t_n)$. Additionally, we have proven that z_n converges to the solution $z(t)$ of the IVP (3.4.4). Uniqueness of the solutions of (3.4.4) and (3.4.1b) imply that $z(t) = y(t)$. This is impossible unless the consistency condition (3.4.3) is satisfied. □

Global error bounds for general one-step methods (3.4.1) have the same form that we saw in Chapter 2 for Euler's method. Thus, a method of order p will converge globally as $O(h^p)$.

Theorem 3.4.3. *Let ϕ satisfy the conditions of Theorem 3.4.2 and let the one-step method be of order p. Then, the global error $e_n = y(t_n) - y_n$ is bounded by*

$$|e_n| \leq \frac{Ch^p}{L}(e^{LT} - 1). \tag{3.4.11}$$

Proof. Since the one-step method is of order p, there exists a positive constant C such that the local error d_n satisfies

$$|d_n| \leq Ch^{p+1}.$$

The remainder of the proof follows the lines of Theorem 2.1.1. □

Problems

1. Prove Theorem 3.4.3.

3.5 Implementation: Error and Step Size Control

We would like to design software that automatically adjusts the step size so that some measure of the error, ideally the global error, is less than a prescribed tolerance. While automatic variation of the step size is easy with one-step methods, it is very difficult to compute global error measures. *A priori* bounds, such as (3.4.11), tend to be too conservative and, hence, use very small step sizes (cf. [16], Section II.3). Other more accurate procedures (cf. [15], pp. 13-14) tend to be computationally expensive. Controlling a

measure of the local (or local discretization) error, on the other hand, is fairly straight forward and this is the approach that we shall study in this section.

A pseudo-code segment illustrating the structure of a one-step method

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h\boldsymbol{\phi}(t_{n-1}, \mathbf{y}_{n-1}, h) \tag{3.5.1a}$$

that performs a single integration step of the vector IVP

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(0) = \mathbf{y}_0, \tag{3.5.1b}$$

is shown in Figure 3.5.1. On input, $\mathbf{y}$ contains an approximation of the solution at time t. On output, t is replaced by $t + h$ and $\mathbf{y}$ contains the computed approximate solution at $t + h$. The step size must be defined on input, but may be modified each time the computed error measure fails to satisfy the prescribed tolerance ϵ.

procedure onestep (**f: vector function**; ϵ**: real**; **var** t, h**: real**; **var y: vector**);

begin
  repeat
    Integrate (3.5.1b) from t to $t + h$ using (3.5.1a);
    Compute $error_{measure}$ at $t + h$;
    if $error_{measure} > \epsilon$ **then** Calculate a new step size h;
  until $error_{measure} \leq \epsilon$;
  $t = t + h$;
  Suggest a step size h for the next step
end;

Figure 3.5.1: Pseudo-code segment of a one-step numerical method with error control and automatic step size adjustment.

In addition to supplying a one-step method, the procedure presented in Figure 3.5.1 will require routines to compute an error measure and to vary the step size. We'll concentrate on the error measure first.

Example 3.5.1. Let us calculate an estimate of the local discretization error of the midpoint rule predictor-corrector. We do this by subtracting the Taylor Taylor's series expansion of the exact solution (3.1.2, 3.1.3) from the expansion of the Runge-Kutta formula (3.1.7) with $a = c = 1/2$, $b_1 = 0$, and $b_2 = 1$. The result is

$$d_n = \frac{h^3}{6}[3(f_{tt} + 2ff_{ty} + f^2 f_{yy}) - (f_t f_y + f f_y^2)]_{(t_{n-1}, y(t_{n-1}))} + O(h^4).$$

Clearly this is too complicated to be used as a practical error estimation scheme.

Two practical approaches to estimating the local and local discretization errors of Runge-Kutta methods are (i) Richardson's extrapolation (or step doubling) and (ii) embedding. We'll study Richardson's extrapolation first.

For simplicity, consider a scalar one-step method of order p having the following form and local error

$$y_n = y_{n-1} + h\phi(t_{n-1}, y_{n-1}, h), \tag{3.5.2a}$$

$$d_n = C_n h^{p+1} + O(h^{p+2}). \tag{3.5.2b}$$

The coefficient C_n may depend on t_{n-1} and $y(t_{n-1})$ but is independent of h. Typically, C_n is proportional to $y^{(p+1)}(t_{n-1})$. Of course, the ODE solution must have derivatives of order $p+2$ for this formula to exist.

Let y_n^h be the solution obtained from (3.5.2a) using a step size h. Calculate a second solution $y_n^{h/2}$ at $t = t_n$ using two steps with a step size $h/2$ and an "initial condition" of y_{n-1} at t_{n-1}. (We'll refer to the solution computed at $t_{n-1/2} = t_{n-1} + h/2$ as $y_{n-1/2}^{h/2}$. Assuming that the error after two steps of size $h/2$ is twice that after one step (i.e., $C_{n-1/2} \approx C_n$), the local errors of both solutions are

$$y_n^h - y(t_n) = C_n h^{p+1} + O(h^{p+2})$$

and

$$y_n^{h/2} - y(t_n) = 2C_n(h/2)^{p+1} + O(h^{p+2})$$

Subtracting the two solutions to eliminate the exact solution gives

$$y_n^h - y_n^h/2 = C_n h^{p+1}(1 - 2^{-p}) + O(h^{p+2}).$$

Neglecting the $O(h^{p+2})$ term, we estimate the local error in the solution of (3.5.2a) as

$$|d_n| \approx |C_n| h^{p+1} = \frac{|y_n^h - y_n^{h/2}|}{1 - 2^{-p}}. \tag{3.5.3a}$$

Computation of the error estimate requires $2s$ additional function evaluations (to compute $y_{n-1/2}$ and y_n) for an s-stage Runge-Kutta method. If $s \approx p$ then approximately

$2p$ extra function evaluations (for scalar systems). This cost for m-dimensional vector problems is approximately $2pm$ function evaluations per step. Richardson's extrapolation is particularly expensive when used with implicit methods because the change of step size requires another Jacobian evaluation and (possible) factorization. It may, however, be useful with DIRK methods because of their lower triangular coefficient matrices.

It's possible to estimate the error of the solution $y_n^{h/2}$ as

$$|d_n^{h/2}| \approx \frac{|C_n|h^{p+1}}{2^p} = \frac{|y_n^h - y_n^{h/2}|}{2^p - 1}. \tag{3.5.3b}$$

Proceeding in this manner seems better than accepting y_n^h as the solution; however, it is a bit risky since we do not have an estimate of the error of the intermediate solution $y_{n-1/2}^{h/2}$.

Finally, the local error estimate (3.5.3a) or (3.5.3b) may be added to y_n^h or $y_n^{h/2}$, respectively, to obtain a higher-order method. For example, using (3.5.3b),

$$y(t_n) = y_n^{h/2} + \frac{y_n^h - y_n^{h/2}}{2^p - 1} + O(h^{p+2}).$$

Thus, we could accept

$$\hat{y}_n^{h/2} = y_n^{h/2} + \frac{y_n^h - y_n^{h/2}}{2^p - 1}$$

as an $O(h^{p+2})$ approximation of $y(t_n)$. This technique, called *local extrapolation*, is also a bit risky since we do not have an error estimate of $\hat{y}_n^{h/2}$. We'll return to this topic in Chapter 4.

Embedding, the second popular means of estimating local (or local discretization) errors, involves using two one-step methods having different orders. Thus, consider calculating two solutions using the p *th*- and $p+1$ *st*-order methods

$$y_n^p = y_{n-1} + h\phi_p(t_{n-1}, y_{n-1}, h), \qquad d_n^p = C_n^p h^{p+1} \tag{3.5.4a}$$

and

$$y_n^{p+1} = y_{n-1} + h\phi_{p+1}(t_{n-1}, y_{n-1}, h), \qquad d_n^{p+1} = C_n^{p+1} h^{p+2}. \tag{3.5.4b}$$

(The superscripts on y_n and d_n are added to distinguish solutions of different order.) The local error of the p-order solution is

$$|d_n^p| = |y_n^p - y(t_n)| = |y_n^p - y_n^{p+1} + y_n^{p+1} - y(t_n)|.$$

Using the triangular inequality

$$|d_n^p| \le |y_n^p - y_n^{p+1}| + |y_n^{p+1} - y(t_n)|.$$

The last term on the right is the local error of the order $p+1$ method (3.5.4b) and is $O(h^{p+2})$; thus,

$$|d_n^p| \le |y_n^p - y_n^{p+1}| + |d_n^{p+1}|.$$

The higher-order error term on the right may be neglected to get an error estimate of the form

$$|d_n^p| \approx |y_n^p - y_n^{p+1}|. \tag{3.5.5}$$

Embedding, like Richardson's extrapolation, is also an expensive way of estimating errors. If the number of Runge-Kutta stages $s \approx p$, then embedding requires approximately $m(p+1)$ additional function evaluations per step for a system of m ODEs.

The number of function evaluations can be substantially reduced by embedding the $p\,th$-order method within an $(s+1)$-stage method of order $p+1$. For explicit Runge-Kutta methods, the tableau of the $(s+1)$-stage method would have the form

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_{s+1} & a_{s+1,1} & a_{s+1,2} & \cdots & a_{s+1,s} & \\ \hline & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_s & \hat{b}_{s+1} \end{array}$$

(Zero's on an above the diagonal in $\mathbf{A}$ are not shown.) Assuming that the $p\,th$-order Runge-Kutta method has s stages, it would be required to have the form

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s+1} & b_s \end{array}$$

With this form, only one additional function evaluation is needed to estimate the error in the (lower) $p\,th$-order method. However, the derivation of such formula pairs is

not simple since the order conditions are nonlinear. Additionally, it may be impossible to obtain a $p+1$-order method by adding a single stage to an s-stage method. Formulas, nevertheless, exist.

Example 3.5.2. The forward Euler method is embedded in the trapezoidal rule predictor-corrector method. The tableaux for these methods are

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

The two methods are

$$k_1 = f(t_{n-1}, y_{n-1}), \qquad k_2 = f(t_{n-1} + h, y_{n-1} + hk_1)$$

$$y_n^1 = y_{n-1} + hk_1$$

$$y_n^2 = y_{n-1} + \frac{h}{2}(k_1 + k_2).$$

Example 3.5.3. There is a three-stage, second-order method embedded in the classical fourth-order Runge-Kutta method. Their tableaux are

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1/2 & 0 & 1/2 & \\ \hline & 0 & 0 & 1 \end{array}$$

These formulas are

$$k_1 = f(t_{n-1}, y_{n-1}), \qquad k_2 = f(t_{n-1} + h/2, y_{n-1} + hk_1/2),$$

$$k_3 = f(t_{n-1} + h/2, y_{n-1} + hk_2/2), \qquad k_4 = f(t_{n-1} + h, y_{n-1} + hk_3),$$

$$y_n^2 = y_{n-1} + hk_3,$$

$$y_n^4 = y_{n-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Example 3.5.4. Fehlberg [14] constructed pairs of explicit Runge-Kutta formulas for non-stiff problems. His fourth- and fifth-order formula pair is

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0
$\hat{}$	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

The $\hat{}$ denotes the coefficients in the higher fifth-order formula. Thus, after determining k_i, $i = 1, 2, \ldots, 6$, the solutions are calculated as

$$y_n^4 = y_{n-1} + h[\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5]$$

and

$$y_n^5 = y_{n-1} + h[\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6].$$

Hairer *et al.* [16], Section II.4 give several Fehlberg formulas. Their fourth- and fifth-order pair is slightly different than the one presented here.

Example 3.5.5. Dormand and Prince [12] develop another fourth- and fifth-order pair that has been designed to minimize the error coefficient of the higher-order method so that it may be used with local extrapolation. Its tableau follows.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$\hat{}$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Having procedures for estimating local (or local discretization) errors, we need to develop practical methods of using them to control step sizes. This will involve the selection of an appropriate (i) error measure, (ii) error test, and (iii) refinement strategy. As indicated in Figure 3.5.1, we will concentrate on step changing algorithms without changing the order of the method. Techniques that automatically vary the order of the method with the step size are more difficult and are not generally used with Runge-Kutta methods (cf., however, Moore and Flaherty [20]).

For vector IVPs (3.5.1b), we will measure the "size" of the solution or error estimate by using a vector norm. Many such metrics are possible. Some that suit our needs are

1. the maximum norm

$$\|\mathbf{y}(t)\|_\infty = \max_{1\le i\le m} |y_i(t)|, \tag{3.5.6a}$$

2. the L_1 or sum norm

$$\|\mathbf{y}(t)\|_1 = \sum_{i=1}^{m} |y_i(t)|, \tag{3.5.6b}$$

3. and the L_2 or Euclidean norm

$$\|\mathbf{y}(t)\|_2 = \left[\sum_{i=1}^{m} |y_i(t)|^2\right]^{1/2}. \tag{3.5.6c}$$

The two most common error tests are control of the *absolute* and *relative* errors. An absolute error test would specify that the chosen measure of the local error be less than a prescribed tolerance; thus,

$$\|\tilde{\mathbf{d}}_n\| \leq \epsilon_A,$$

where the $\tilde{}$ signifies the local error estimate rather than the actual error. Using a relative error test, we would control the error measure relative to the magnitude of the solution, e.g.,

$$\|\tilde{\mathbf{d}}_n\| \leq \epsilon_R \|\mathbf{y}_n\|.$$

It is also common to base an error test on a combination of an absolute and a relative tolerance, i.e.,

$$\|\tilde{\mathbf{d}}_n\| \leq \epsilon_R \|\mathbf{y}_n\| + \epsilon_A.$$

When some components of the solution are more important than others it may be appropriate to use a weighted norm with $y_i(t)$ in (3.5.6) replaced by $y_i(t)/w_i$, where

$$\mathbf{w} = [w_1, w_2, \ldots, w_m]^T \tag{3.5.7a}$$

is a vector of positive weights. As an example, consider the weighted maximum norm of the local error estimate

$$\|\tilde{\mathbf{d}}_n\|_{\mathbf{w},\infty} = \max_{1 \leq i \leq m} \left| \frac{\tilde{d}_{n,i}}{w_i} \right|, \tag{3.5.7b}$$

where $\tilde{d}_{n,i}$ denotes the local error estimate of the $i\ th$ component of $\mathbf{d}_n$.

Use of a weighted test such as

$$\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}} \leq \epsilon \tag{3.5.7c}$$

adds flexibility to the software. Users may assign weights prior to the integration in proportion to the importance of a variable. The weighted norm may also be used to simulate a variety of standard tests. Thus, for example, an absolute error test would be obtained by setting $w_i = 1$, $i = 1, 2, \ldots, m$, and $\epsilon = \epsilon_A$. A mixed error test where the integration step is accepted if the local error estimate of the $i\ th$ ODE does not exceed

$$\epsilon_R |y_{n,i}| + \epsilon_A$$

may be specified by using the maximum norm and selecting

$$\epsilon = \max(\epsilon_A, \epsilon_R)$$

and

$$w_i = (\epsilon_R |y_{n,i}| + \epsilon_A)/\epsilon.$$

Present Runge-Kutta software controls:

1. the local error

$$\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}} \leq \epsilon, \tag{3.5.8a}$$

2. the local error per unit step

$$\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}} \leq h\epsilon, \tag{3.5.8b}$$

3. or the indirect (extrapolated) local error per unit step

$$\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}} \leq Ch\epsilon, \tag{3.5.8c}$$

   where C is a constant depending on the method.

The latter two formulas are attempts to control a measure of the global error.

Let us describe a step size selection process for controlling the local error per unit step in a $p\ th$ order Runge-Kutta method. Suppose that we have just completed an integration from t_{n-1} to t_n. We have computed an estimate of the local error $\tilde{\mathbf{d}}_n$ using either Richardson's extrapolation or order embedding. We compare $\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}}$ with the prescribed tolerance and

1. if $\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}} > \epsilon$ we reject the step and repeat the integration with a smaller step size,

2. otherwise we accept the step and suggest a step size for the subsequent step.

In either case,

$$\frac{\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}}}{h} \approx C_n h^p.$$

Ideally, we would like to compute a step size h_{OPT} so that

$$\epsilon \approx C_n h_{OPT}^p.$$

Eliminating the coefficient C_n between the two equations

$$\frac{\epsilon}{\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}}} \approx \frac{h_{OPT}^p}{h^{p+1}},$$

or

$$\frac{h_{OPT}}{h} \approx \left(\frac{h\epsilon}{\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}}}\right)^{1/p}. \tag{3.5.9a}$$

The error estimates are based upon an asymptotic analysis and are, thus, not completely reliable. Therefore, it is best to include safety factors such as

$$h_{OPT} = h \min\left\{\eta_{MAX}, \max\left[\eta_{MIN}, \eta_s \left(\frac{\epsilon}{\|\tilde{\mathbf{d}}_n\|_{\mathbf{w}}}\right)^{1/p}\right]\right\}. \tag{3.5.9b}$$

The factors η_{MAX} and η_{MIN} limit the maximum step size increase and decrease, respectively, while η_s tends to make step size changes more conservative. Possible choices of the parameters are $\eta_{MAX} = 5$, $\eta_{MIN} = 0.1$, and $\eta_s = 0.9$. Step size control based on either (3.5.8a) or (3.5.8c) works similarly. In general, the user must also provide a maximum step size h_{MAX} so that the code does not miss interesting features in the solution.

Selection of the initial step size is typically left to the user. This can be somewhat problematical and several automatic initial step size procedures are under investigation. One automatic procedure that seems to be reasonably robust is to select the initial step size as

$$h = \left(\frac{\epsilon}{1/T^{p^*} + \|\mathbf{f}(0, \mathbf{y}(0))\|^{p^*}}\right)^{1/p^*},$$

where T is the final time and $p^* = p + 1$ for local error control and $p^* = p$ for local error per unit step control.

Example 3.5.6. ([16], Section II.4). We report results when several explicit fourth-order explicit Runge-Kutta codes were applied to

$$y_1' = 2ty_1 \log(\max(y_2, 10^-3)), \qquad y_1(0) = 1,$$

$$y_2' = -2ty_2 \log(\max(y_1, 10^-3)), \qquad y_2(0) = e.$$

0							
$\frac{1}{2}$	$\frac{1}{2}$						
$\frac{2}{3}$	$\frac{2}{9}$	$\frac{4}{9}$					
$\frac{1}{3}$	$\frac{7}{36}$	$\frac{2}{9}$	$-\frac{1}{12}$				
$\frac{5}{6}$	$-\frac{35}{144}$	$-\frac{55}{36}$	$\frac{35}{48}$	$\frac{15}{8}$			
$\frac{1}{6}$	$-\frac{1}{360}$	$-\frac{11}{36}$	$-\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{10}$		
1	$-\frac{41}{260}$	$\frac{22}{13}$	$\frac{43}{156}$	$-\frac{118}{39}$	$\frac{32}{195}$	$\frac{80}{39}$	
	$\frac{13}{200}$	0	$\frac{11}{40}$	$\frac{11}{40}$	$\frac{4}{25}$	$\frac{4}{25}$	$\frac{13}{200}$

Table 3.5.1: Butcher's seven-stage sixth-order explicit Runge-Kutta method.

The exact solution of this problem is

$$y_1(t) = e^{\sin t^2}, \qquad y_2(t) = e^{\cos t^2}.$$

Hairer *et al.* [16] solved the problem on $0 \leq t \leq 5$ using tolerances ranging from 10^{-7} to 10^{-3}. The results presented in Figure 3.5.2 compare the base 10 logarithms of the maximum global error and the number of function evaluations.

The several methods that are not identified in Figure 3.5.2 are the more traditional formulas, including the classical Runge-Kutta method (solid line). All of these are listed in Hairer *et al.* [16], Section II.1. "Fehlberg's method" is the fourth- and fifth-order pair given in Example 3.5.4. The "Dormand-Prince" method is the fourth- and fifth-order pair of Example 3.5.5. "Butcher's method" is the sixth-order seven-stage formula shown in Table 3.5.1. It is the only formula that is beyond fourth or fifth order. Results in the lower graph of Figure 3.5.2 use local extrapolation; thus, the higher-order solution of the pair is kept, even though it has no local error estimate.

Of all the methods shown in Figure 3.5.2, the Dormand-Prince and Fehlberg methods appear to have the greatest accuracy for a given cost. The higher-order Butcher formula gains appeal as accuracy increases. The Dormand-Prince method has a distinct advantage relative to the Fehlberg method when local extrapolation is used. As noted, the Dormand-Prince method was designed for this purpose. For this problem, which has a smooth

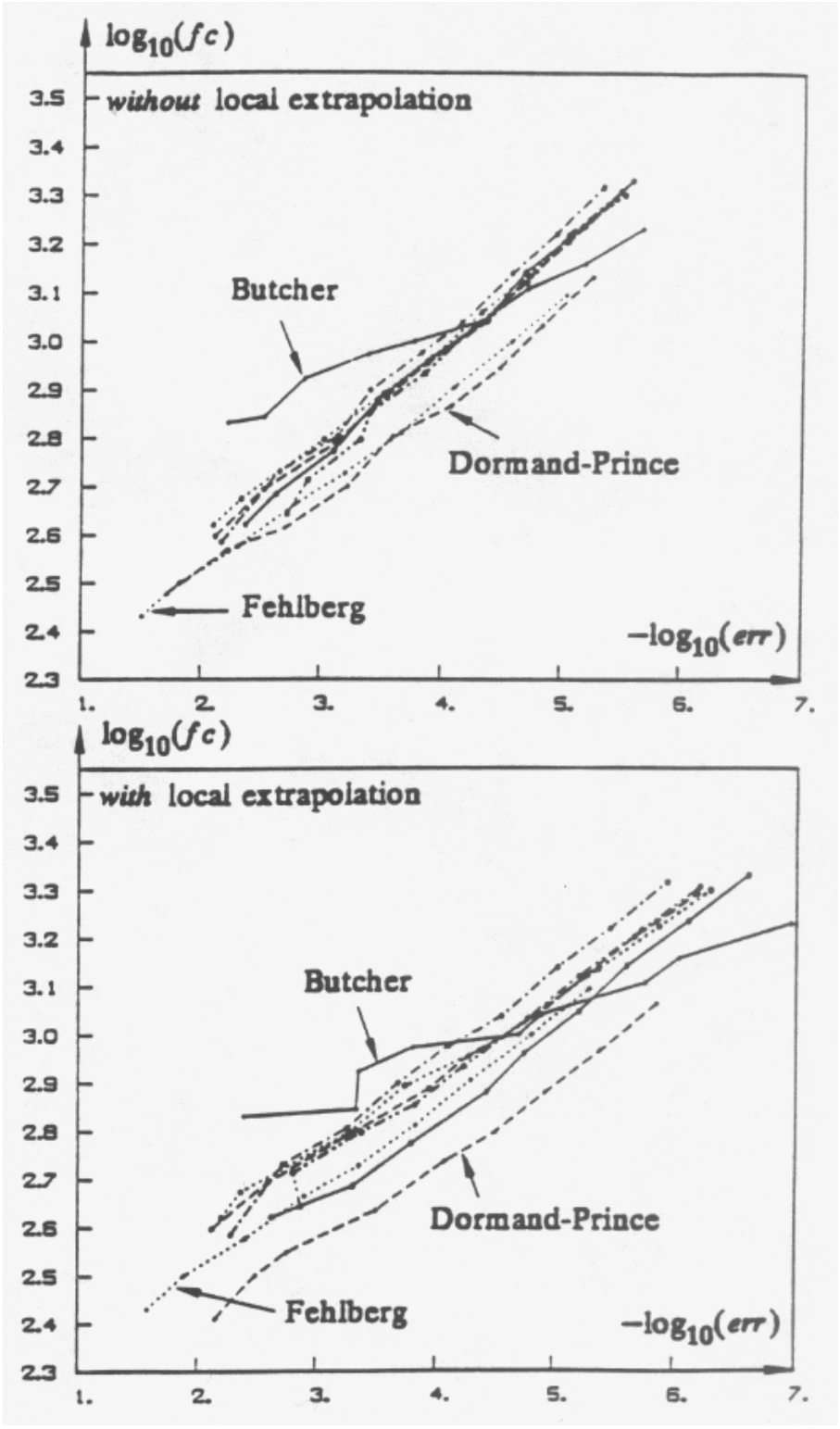

Figure 3.5.2: Accuracy vs. effort for several explicit Runge-Kutta methods [16].

solution, the Dormand-Prince method is nearly a factor of 10 more accurate with local extrapolation than without it.

Numerous other implementation details have not been addressed. These include (i) Newton iteration for implicit systems and (ii) portability. We will return to these topics when discussing multistep methods in Chapter 5. While we have not discussed competing methods, we'll nevertheless conclude that explicit Runge-Kutta formulas should be considered when

1. the functions defining the differential system are simple or
2. the solution has discontinuities.

The latter case is interesting. Discontinuities can be ignored and treated with the automatic step-size selection procedures. This is easier to do with Runge-Kutta methods than with competing approaches such as multistep methods. Explicit treatment of discontinuities by locating the discontinuity and restarting the solution there is also simpler with Runge-Kutta methods than with competing methods.

Implicit Runge-Kutta methods are useful when high accuracy and A- or L-stability are needed simultaneously. This occurs with problems where $\mathbf{f}_{\mathbf{y}}(t, \mathbf{y})$ has eigenvalues with large (neagative) real or imaginary parts. We will postpone a comparison of methods for these problems until examining multistep methods in Chapter 5. At this time, we'll note that software based on fifth-, seventh- and ninth-order Radau methods [17] has done extremely well when solving stiff IVPs. The STRIDE software [6] based on SIRK methods has been successful, but less so than the Radau methods.

Problems

1. The aim of this problem is to write a subroutine or procedure for performing one step of a fourth-order variable step Runge-Kutta method applied to vector IVPs of the form

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0.$$

   1.1. Write a subroutine or procedure to perform one step of a fourth-order explicit Runge-Kutta method. You may use the classical, Fehlberg, Dormand and Prince, or other formula as long as it is fourth order.

1.2. Test your procedure using fixed step integration with step sizes $h = 1/2$, $1/4$, $1/8$, ... using the test IVPs

$$y' = y, \qquad y(0) = 1, \qquad 0 \leq\leq 1$$

and

$$\frac{d^2y}{dt^2} + 2k\frac{dy}{dt} + n^2y = 0, \qquad 0 < t < 3,$$

$$y(0) = 1, \qquad dyoverdt(0) = 0.$$

For the second example use $n = 10$, $k = 6$ and $n = 10$, $k = 8$. In each case, present results (tables and/or graphs) of the global error and number of function evaluations as functions of h. Estimate the rate of convergence of the method and compare it with the theoretical rate.

1.3. Replace the fixed step size strategy above with a variable step size technique of your choice. Base step size selection on control of the local error, which may be estimated using either step doubling or embedding. Compare the performance of your code on the above problems with the fixed step performance.

Bibliography

[1] M. Abromowitz and I. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1995.

[2] R. Alexander. Diagonally implicit runge-kutta methods for stiff o.d.e.'s. *SIAM J. Numer. Anal.*, 14:1006–1021, 1977.

[3] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998.

[4] R.L. Burden and J.D. Faires. *Numerical Analysis*. PWS-Kent, Boston, fifth edition, 1993.

[5] K. Burrage. A special family of runge-kutta methods for solving stiff differential equations. *BIT*, 18:22 – 41, 1978.

[6] K. Burrage, J.C. Butcher, and R.H. Chipman. An implementation of singly-implicit runge-kutta methods. *BIT*, 20:326 – 340, 1980.

[7] J.C. Butcher. Implicit runge-kutta processes. *Maths. Comp.*, 18:50–64, 1964.

[8] J.C. Butcher. On the attainable order of runge-kutta methods. *Maths. Comp.*, 19:408 – 417, 1965.

[9] J.C. Butcher. On the implementation of implicit runge-kutta methods. *BIT*, 16:237 – 240, 1976.

[10] J.C. Butcher. A transformed implicit runge-kutta method. *J. Assoc. Comput. Mach.*, 26:731 – 738, 1979.

[11] J.C. Butcher. *The Numerical Analysis of Ordinary Differential Equations.* John Wiley and Sons, New York, 1987.

[12] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *J. Comput. Appl. Math.*, 6:19 – 26, 1980.

[13] B.L. Ehle. High order a-stable methods for numerical solution of systems of differential equations. *BIT*, 8:276–278, 1968.

[14] E. Fehlberg. Klassische runge-kutta formeln vierter und niedrigerer ordnung mit schrittweiten-kontrolle und ihre andwendung auf warmeleitungs-probleme. *Computing*, 6:61–71, 1970.

[15] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice Hall, Englewood Cliffs, 1971.

[16] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, Berlin, second edition, 1993.

[17] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems.* Springer-Verlag, Berlin, 1991.

[18] P.C. Hammer and J.W. Hollingsworth. Trapezoidal methods of approximating solutions of differential equations. *Math. Tables Aids Comp.*, 9:92–96, 1955.

[19] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods.* John Wiley and Sons, New York, 1966.

[20] P.K. Moore and J.E. Flaherty. High-order adaptive finite element-singly implicit runga-kutta methods for parabolic differential equations. *BIT*, 33:309–331, 1993.

[21] C. Runge. Uber die numerishce aufl osung von differentialgleichungen. *Math. Ann.*, 46:167 – 178, 1895.

Chapter 4

Extrapolation Methods

4.1 Polynomial and Rational Extrapolation

In Chapter 3, we used extrapolation as a tool to estimate the discretization errors of high-order Runge-Kutta methods. We learned that these error estimates are "asymptotically correct;" thus, they converge at the same rate as actual discretization errors. Indeed, using local extrapolation, we added these error estimates to the computed solution to obtain a higher-order approximation. In this chapter, we study the use of low-order methods with repeated extrapolation to obtain high-order methods satisfying prescribed accuracy criteria. As in previous chapters, we'll consider the scalar IVP

$$y' = f(t, y), \qquad y(0) = y_0. \tag{4.1.1}$$

With the complications introduced by computing solutions on meshes having different spacings, let us change our notation for the numerical solutions and let $z(t, h)$ denote the numerical approximation of $y(t)$ at time t obtained using step size h. If $t_n = nh$, then $y_n = z(t_n, h)$.

Suppose that the global error of a numerical method has an expansion in powers of h of the form

$$z(t, h) = y(t) + \sum_{i=1}^{q} c_i(t)h^i + O(h^{q+1}), \tag{4.1.2}$$

with $c_j(t) = 0$, $j = 1, 2, \ldots, p-1$, for a method of order $p < q$. Gragg [4] showed that every explicit one-step method has such an expansion when $y(t)$ is smooth. Implicit Runge-Kutta methods may also have expansions of this form.

The idea of extrapolation is to combine solutions $z(t, h_j)$ obtained with different step sizes h_j, $j = 0, 1, \ldots$, so that successive terms of the error expansion (4.1.2) are eliminated; thus, resulting in a higher-order approximation. The process is commonly called *Richardson's extrapolation*. Let us begin with a simple example.

Example 4.1.1. If solutions are smooth, the global error of either the forward or backward Euler methods have series expansions of the form

$$z(t, h) = y(t) + c_1 h + c_2 h^2 + \ldots .$$

Obtain two solutions at the same time t using step sizes of h_0 and $h_0/2$, i.e.,

$$z(t, h_0) = y(t) + c_1 h_0 + c_2 h_0^2 + \ldots$$

$$z(t, h_0/2) = y(t) + c_1 \frac{h_0}{2} + c_2 (\frac{h_0}{2})^2 + \ldots .$$

Subtracting these two equations, we eliminate $y(t)$ and obtain

$$\frac{c_1 h_0}{2} = z(t, h_0) - z(t, h_0/2) - \frac{3c_2}{4} h_0^2 + \ldots .$$

The leading term of this expression furnishes a discretization error estimate of either solution. Substituting the above expression into either of the two global error expansions yields

$$2z(t, h_0/2) - z(t, h_0) = y(t) - \frac{c_2}{2} h_0^2 + \ldots .$$

Thus, $2z(t, h_0/2) - z(t, h_0)$ provides a higher-order ($O(h^2)$) approximation of $y(t)$ than either $z(t, h_0)$ or $z(t, h_0/2)$.

The same result can be obtained by approximating $z(t, h)$ by a linear polynomial $R_1(t, h)$ that interpolates $z(t, h)$ at $h = h_0$ and $h_0/2$. Thus, let

$$R_1(t, h) = \alpha_0(t) + \alpha_1(t) h.$$

The interpolation conditions require

$$z(t, h_0) = \alpha_0(t) + \alpha_1(t) h_0$$

and

$$z(t, h_0/2) = \alpha_0(t) + \frac{\alpha_1(t) h_0}{2}.$$

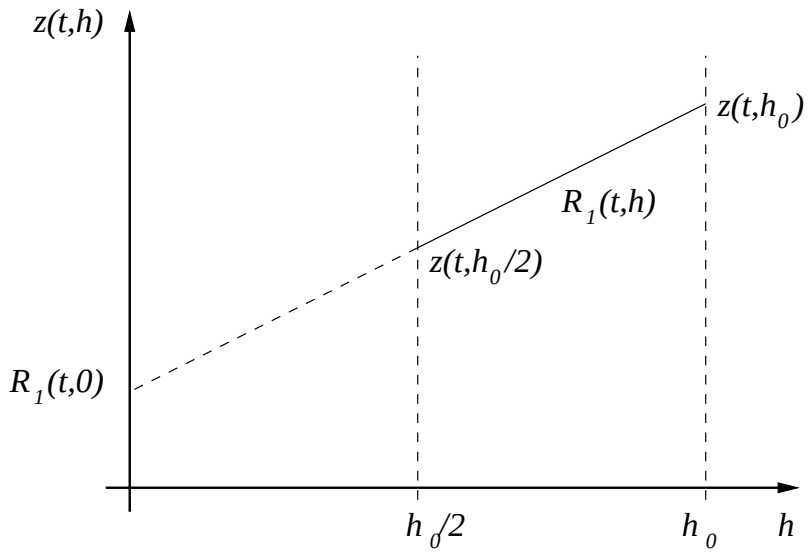

Figure 4.1.1: Interpretation of Richardson's extrapolation for Example 4.1.1.

Thus,

$$\alpha_0 = 2z(t, h_0/2) - z(t, h_0), \qquad \alpha_1 = \frac{2}{h_0}[z(t, h_0/2) - z(t, h_0)].$$

As shown in Figure 4.1.1, the higher-order solution is obtained as

$$R_1(t, 0) = \alpha_0 = 2z(t, h_0/2) - z(t, h_0).$$

The general extrapolation procedure consists of generating a sequence of solutions $z(t, h_j)$, $j = 0, 1, \ldots, q$, with $h_0 > h_1 > \ldots > h_q$ and interpolating these solutions by an $q\ th$ degree polynomial $R_q^0(t, h)$. The desired higher-order approximation is the value of the interpolating polynomial at $h = 0$, i.e., $R_q^0(t, 0)$. The meaning of the superscript 0 will become clear shortly.

The *Aitken-Neville algorithm* provides a simple way of generating the necessary approximations in the form a recurrence relation where higher-order solutions are obtained from lower-order ones. Suppressing the t dependence, let $R_q^i(h)$ be the unique polynomial of degree q that satisfies the interpolation conditions

$$R_q^i(h) = z(t, h_j), \qquad j = i, i + 1, \ldots, i + q. \tag{4.1.3}$$

Consider $R_q^i(h)$ in the form

$$R_q^i(h) = \Phi_{iq}(h) R_{q-1}^i(h) + (1 - \Phi_{iq}(h)) R_{q-1}^{i+1}(h) \tag{4.1.4a}$$

where $\Phi_{iq}(h)$ is a linear polynomial in h. By assumption,

$$R^i_{q-1}(h_j) = z(t, h_j), \qquad j = i, i+1, \ldots, i+q-1,$$

and

$$R^{i+1}_{q-1}(h_j) = z(t, h_j), \qquad j = i+1, i+2, \ldots, i+q;$$

thus, the required interpolation conditions

$$R^i_q(h) = z(t, h_j), \qquad j = i+1, i+2, \ldots, i+q-1,$$

are satisfied for all choices of $\Phi_{iq}(h)$. If we additionally select

$$\Phi_{iq}(h_i) = 1, \qquad \Phi_{iq}(h_{i+q}) = 0,$$

then

$$R^i_q(h_i) = R^i_{q-1}(h_i) = z(t, h_i), \qquad R^i_q(h_{i+q}) = R^{i+1}_{q-1}(h_{i+q}) = z(t, h_{i+q}).$$

Since $\Phi_{iq}(h)$ is a linear function of h then

$$\Phi_{iq}(h) = \frac{h - h_{i+q}}{h_i - h_{i+q}}. \tag{4.1.4b}$$

Combining (4.1.4a) and (4.1.4b)

$$R^i_q(h) = \frac{(h - h_{i+q})R^i_{q-1}(h) + (h_i - h)R^{i+1}_{q-1}(h)}{h_i - h_{i+q}}. \tag{4.1.5}$$

We'll denote the extrapolated solutions $R^i_q(0)$ simply as R^i_q. These functions may be simply generated in a tableau as indicated in Figure 4.1.2. Using (4.1.5), the entries of the tableau are

$$\begin{array}{llllll}
z(t,h_0) = R^0_0 & R^0_1 & R^0_2 & R^0_3 \\
z(t,h_1) = R^1_0 & R^1_1 & R^1_2 & \vdots \\
z(t,h_2) = R^2_0 & R^2_1 & \vdots & \\
z(t,h_3) = R^3_0 & \vdots & & \\
\vdots &&&
\end{array}$$

Figure 4.1.2: Tableau for Richardson's extrapolation.

$$R_q^i = R_q^i(0) = \frac{h_i R_{q-1}^{i+1} - h_{i+q} R_{q-1}^i}{h_i - h_{i+q}} = R_{q-1}^{i+1} + \frac{R_{q-1}^{i+1} - R_{q-1}^i}{(\frac{h_i}{h_{i+q}}) - 1}. \tag{4.1.6}$$

If this extrapolation technique is used with a first-order method, then the values of R_q^i increase in accuracy as either i or q increase, provided that $y(t)$ is smooth. In this case, the global error $y(t) - R_q^i$ is $O(h^{q+1})$, provided that $h_i \to 0$ as $i \to \infty$ [4]. Finally, we note that it is not necessary to halve the step size after each pass. Other step size sequences can be used to reduce computation. For example

$$\{h_0,\ h_0/2,\ h_0/3,\ h_0/4,\ h_0/6,\ h_0/8,\ \ldots\}.$$

This should be done carefully since some sequences lead to a loss of stability ([5], Section II.9).

Example 4.1.2. Consider the solution of

$$y' = -y, \qquad 0 < t \leq 1, \qquad y(0) = 1,$$

using the forward Euler method with step sizes $h_i = 2^{-i}$, $i = 0, 1, \ldots, 8$. The results at $t = 1$ are reported in Table 4.1.1. The entries in the first column ($q = 0$) are computed by Euler's method with step size h_i. Subsequent columns are obtained using (4.1.6). Let us verify the entry in the first row and second column of the upper table by using (4.1.6) with $i = 0$ and $q = 1$; thus,

$$R_1^0 = R_0^1 + \frac{R_0^1 - R_0^0}{(\frac{h_0}{h_1}) - 1} = 0.25 + \frac{0.25 - 0.0}{2.0 - 1.0} = 0.5.$$

Columns converge at increasing powers of h; thus, errors $(y(1) - R_q^i)$ decrease as $O(h)$ in the first ($q = 0$) column, as $O(h^2)$ in the second ($q = 1$) column, *etc.* This is confirmed in the bottom table, which consists of the errors divided by h_i^{q+1}.

Increasing q or i increases the number of calculations. If the basic integration scheme is Euler's method, computations increase at the rate of 2^{q+i} when the basic step $h_0 = 1$ is repeatedly halved. If worst-case round-off occurs at each step, the answers would lose one bit of precision for each increase in i or q. We could reduce this problem by using a different strategy to choose step sizes. For example, select $h_0 = 1$, $h_1 = 1/r$, ...,

	$q=0$	1	2	3	4
$i=0$	0.0000	0.5000	0.3438	0.3701	0.3678
1	0.2500	0.3828	0.3668	0.3679	0.3679
2	0.3164	0.3708	0.3678	0.3679	0.3679
3	0.3436	0.3685	0.3679	0.3679	0.3679
4	0.3561	0.3680	0.3679	0.3679	0.3679
5	0.3621	0.3679	0.3679	0.3679	
6	0.3650	0.3679	0.3679		
7	0.3664	0.3679			
8	0.3672				

	$q=0$	1	2	3	4
$i=0$	3.6788e-01	-1.3212e-01	2.4129e-02	-2.2263e-03	1.0452e-04
1	1.1788e-01	-1.4933e-02	1.0682e-03	-4.1157e-05	8.3834e-07
2	5.1473e-02	-2.9321e-03	9.7508e-05	-1.7864e-06	1.7546e-08
3	2.4271e-02	-6.5990e-04	1.0625e-05	-9.5199e-08	4.6022e-10
4	1.1805e-02	-1.5701e-04	1.2449e-06	-5.5185e-09	1.3240e-11
5	5.8242e-03	-3.8318e-05	1.5078e-07	-3.3249e-10	
6	2.8929e-03	-9.4664e-06	1.8557e-08		
7	1.4417e-03	-2.3527e-06			
8	7.1969e-04				

	$q=0$	1	2	3	4
$i=0$	3.6788e-01	-1.3212e-01	2.4129e-02	-2.2263e-03	1.0452e-04
1	2.3576e-01	-5.9732e-02	8.5453e-03	-6.5851e-04	2.6827e-05
2	2.0589e-01	-4.6914e-02	6.2405e-03	-4.5731e-04	1.7967e-05
3	1.9416e-01	-4.2234e-02	5.4402e-03	-3.8993e-04	1.5080e-05
4	1.8888e-01	-4.0194e-02	5.0990e-03	-3.6166e-04	1.3883e-05
5	1.8637e-01	-3.9238e-02	4.9408e-03	-3.4864e-04	
6	1.8515e-01	-3.8774e-02	4.8645e-03		
7	1.8454e-01	-3.8546e-02			
8	1.8424e-01				

Table 4.1.1: Solution of Example 4.1.2 by Richardson's extrapolation. The upper table presents solution, the middle table presents errors in R_q^i, and the lower table presents errors divided by h_i^{q+1}.

$h_i = 1/r^i$, for $1 < r \leq 2$. The worst case rounding errors in R_0^i will be proportional to r^i, since this is approximately the number of steps. The error in R_1^i could be as large as

$$r^{i+1} + \frac{r^{i+1} + r^i}{r-1} = r^{i+1}(\frac{r+1/r}{r-1}).$$

The error in R_q^i could be as large as

$$r^{i+q}(\frac{r+1/r}{r-1})(\frac{r^2+1/r}{r^2-1})\ldots(\frac{r^q+1/r}{r^q-1}).$$

Values of r near 2 will make the expression r^{i+q} large whereas values of q near unity will make the denominators in the above expression small. Bulirsch and Stoer [2] used sequences with $r = 3/2$ or $r = 4/3$ so that r^{i+q} grows more slowly than with $r = 2$.

If the extrapolation is started with a method of order p ($p \geq 1$) and the method has an expansion in h of the form

$$z(t,h) = y(t) + \sum_{i=0}^{q} c_i(t)h^{p+i} + O(h^{p+q+1}).$$

In this case, the approximation R_q^i has an error of order $O(h^{p+q+1})$.

Some methods have error expansions that only contain even powers of h, e.g.,

$$z(t,h) = y(t) + \sum_{i=1}^{q} c_i(t)h^{i\gamma} + O(h^{(q+1)\gamma}),$$

where γ is typically 2 or 4. The extrapolation algorithm can be modified to take advantage of this by utilizing polynomials in h^γ to obtain

$$R_q^i = R_{q-1}^{i+1} + \frac{R_{q-1}^{i+1} - R_{q-1}^i}{(\frac{h_i}{h_{i+q}})^\gamma - 1}. \tag{4.1.7}$$

Now, the order of the approximation is increased by γ in each successive column of the extrapolation tableau. If the trapezoidal rule is solved exactly at each step, then it has an expansion in powers of h^2 ($\gamma = 2$).

Extrapolation with rational functions is also possible ([5], Section II.9). The basic idea is to approximate $z(t,h)$ by a rational function $R(h)$ and then evaluate $R(0)$. Bulirsch and Stoer [1] derived a scheme where $R_q^i(h)$ is defined as the rational approximation that interpolates $z(t,h)$ at $h = h_i, h_{i+1}, \ldots, h_{i+q}$ for $h = h_i > h_{i+1} > \ldots > h_{i+q}$. The values of $R_q^i = R_q^i(0)$ can be obtained from the following recursion

$$R_{-1}^i = 0, \qquad R_0^i = z(t,h_i), \tag{4.1.8a}$$

$$R_q^i = R_{q-1}^{i+1} + \frac{R_{q-1}^{i+1} - R_{q-1}^i}{(\frac{h_i}{h_{i+q}})^2[1 - R_{q-1}^{i+1} - R_{m-1}^i R_{q-1}^{i+1} - R_{q-2}^{i+2}] - 1}, \tag{4.1.8b}$$

when the method has an error expansion in powers of h^2. The computation of R_q^i according to (4.1.8b) is equivalent to interpolating $z(t,h)$ by a function of the form

$$R_j^i(h) = \begin{cases} \frac{a_0+a_1h^2+\ldots+a_jh^j}{b_0+b_1h^2+\ldots+b_jh^j}, & \text{if } j \text{ is even} \\ \frac{a_0+a_1h^2+\ldots+a_{j-1}h^{j-1}}{b_0+b_1h^2+\ldots+b_{j-1}h^{j-1}}, & \text{if } j \text{ is odd} \end{cases} . \tag{4.1.8c}$$

There are several enhancements to the basic extrapolation approach. Some of these follow.

1. A variable-order extrapolation algorithm could choose the order (i.e., the number of columns in the tableau) adaptively. Error estimates can be computed as the difference between the first subdiagonal element and the diagonal or by the difference between two successive diagonal elements.

2. Extrapolation methods can be written as Runge-Kutta methods if the step size and order of the extrapolation are fixed. In this case, convergence, stability, and error bounds follow from the results of Chapter 3 for general one-step methods.

3. Implicit methods can be used in combination with extrapolation to solve stiff problems. A survey of the state of the art was written by Deuflhard [3] who prefers polynomial extrapolation to rational extrapolation because rational extrapolation lacks translation invariance, rational extrapolation can impose restrictions on the base step size, and polynomial extrapolation is slightly more efficient.

Bibliography

[1] R. Bulirsch and J. Stoer. Fehlerabschätzungen und extrapolation mit rationalen funktionen bei verfahren vom Richardson-typus. *Numer. Math.*, 6:413–427, 1964.

[2] R. Bulirsch and J. Stoer. Numerical treatment of ordinary differential equations by extrapolation methods. *Numer. Math.*, 8:1–13, 1966.

[3] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. *SIAM Review*, 27:505–535, 1985.

[4] W.B. Gragg. On extrapolation algorithms for ordinary initial value problems. *SIAM J. Numer. Anal.*, 2:384–403, 1964.

[5] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, Berlin, second edition, 1993.

Chapter 5

Multivalue or Multistep Methods

5.1 Introduction

One-step methods only require information about the solution at one time, say $t = t_{n-1}$ to compute the solution at an advanced time $t = t_n$. After integrating away from the initial point, we have several solution values that can be used to predict the solution at an advanced point. In this Chapter, we use these past solution values to construct "multistep" methods that potentially have fewer function evaluations per time step than one-step methods. To begin, let us define a *linear multistep method* for the scalar IVP

$$y' = f(t, y), \qquad t > 0, \qquad y(0) = y_0 \tag{5.1.1}$$

as

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = h \sum_{i=0}^{k} \beta_i f_{n-i}, \tag{5.1.2a}$$

where

$$f_j = f(t_j, y_j). \tag{5.1.2b}$$

In applying (5.1.2), we assume that approximate solutions y_{n-1}, y_{n-2}, ..., y_{n-k} are known and that we seek to calculate y_n (Figure 5.1.1). In particular, y_0, y_1, ..., y_{k-1} must be known in order to start using the method. We'll discuss methods for obtaining these "starting values;" however, for the moment, assume that they are calculated by a one-step method. Some other assumptions and observations concerning (5.1.2) follow.

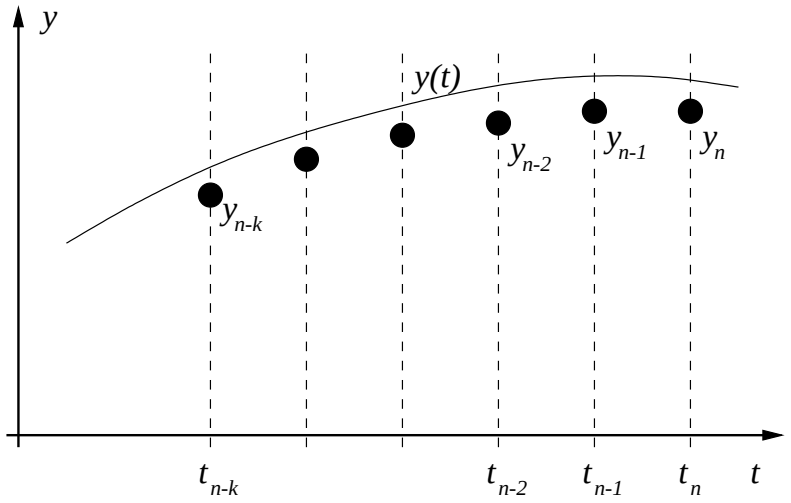

Figure 5.1.1: Domain of a k-step linear multistep method.

1. The step size h has been assumed to be constant for all steps. Designing variable step formulas and changing step sizes will be more complex than with one-step methods. We'll return to this subject later. item We normalize (5.1.2b) by selecting

$$\alpha_0 = 1 \tag{5.1.2c}$$

and also assume that at least one other α_i, $i = 1, 2, \ldots, k$, or β_i, $i = 0, 1, \ldots, k$, is nonzero.

2. If $\beta_0 = 0$ the method is explicit, otherwise it is implicit. For an explicit method, y_n may be directly obtained in terms of y_{n-i} and f_{n-i}, $i = 1, 2, \ldots, k$. An implicit method generally requires the solution of a nonlinear problem.

3. There is a linear relationship between y_{n-i}, $i = 0, 1, \ldots, k$, and f_{n-i}, $i = 0, 1, \ldots, k$. For this reason, (5.1.2) is called a *linear multistep method*, even when $f(t, y)$ is a nonlinear function of y.

We'll discuss systematic approaches to constructing multistep methods in the next two sections; however, let us illustrate the approach using the method of *undetermined coefficients*. With this technique we

1. assume a particular form of the general formula (5.1.2) by, possibly, restricting some of the coefficients and

2. determine the remaining coefficients of (5.1.2) such that they match terms of a Taylor's series expansion of the exact ODE solution to as high a degree as possible. Equivalently, the coefficients can be determined so that (5.1.2) produces the exact ODE solution when $y(t)$ is a polynomial to as high a degree as possible.

Here's an example.

Example 5.1.1. Consider a multistep method of the form

$$y_n + \alpha_1 y_{n-1} + \alpha_2 y_{n-2} = h\beta_1 f_{n-1}.$$

This explicit two-step formula has three undetermined coefficients (α_1, α_2, and β_1) and we'll determine them so that the numerical method is exact when $y(t)$ is an arbitrary quadratic polynomial. Since the multistep method is linear, it suffices to make the formula exact when $y(t)$ is 1, t, and t^2. If $y(t) = 1$ then $f(t, y) = 0$ and (5.1.2) yields

$$1 + \alpha_1 + \alpha_2 = 0. \tag{5.1.3a}$$

When $y(t) = t$, $f(t, y) = 1$ and (5.1.2) yields

$$t_n + \alpha_1(t_n - h) + \alpha_2(t_n - 2h) = h\beta_1.$$

Using (5.1.3a),

$$-\alpha_1 - 2\alpha_2 = \beta_1. \tag{5.1.3b}$$

When $y(t) = t^2$, $f(t, y) = 2t$ and (5.1.2) yields

$$t_n^2 + \alpha_1(t_n - h)^2 + \alpha_2(t_n - 2h)^2 = 2h\beta_1(t_n - h).$$

This may be simplified by (5.1.3a,b) to

$$\alpha_1 + 4\alpha_2 = -2\beta_1. \tag{5.1.3c}$$

The solution of (5.1.3a,b,c) is

$$\alpha_1 = 0, \qquad \alpha_2 = -1, \qquad \beta_1 = 2;$$

thus, the method is

$$y_n = y_{n-2} + 2hf_{n-1}, \qquad n = 2, 3, \ldots . \tag{5.1.4a}$$

This scheme is called the "leap frog" scheme. It has only one function evaluation per step and, once started (with y_0 and y_1), is as simple as the explicit Euler method.

Without having introduced a formal definition of the local discretization error for multistep methods, let's use our experience with one-step methods to define it for the leap frog method as

$$\tau_n = \frac{y(t_n) - y(t_{n-2})}{2h} - f(t_{n-1}, y(t_{n-1})).$$

Since (5.1.4a) is exact when $y(t)$ is a quadratic polynomial, we may either infer or a Taylor's series expansion to show that

$$\tau_n = Ch^2 y'''(\xi_n), \qquad \xi_n \in (t_{n-2}, t_n).$$

The numerical constant C may also be determined by the method of undetermined coefficients. To do this, we select the simplest ODE solution for which the numerical method does not produce an exact solution. In the case of the leap frog method, this would be any cubic polynomial, and we select $y(t) = t^3$. Substitution into the two expressions for the local discretization error yields

$$\frac{t_n^3 - (t_n - 2h)^3}{2h} - 3(t_n - h)^2 = 6Ch^2.$$

The location of the point ξ_n is irrelevant for this exact solution and we find $C = 1/3$. Thus,

$$\tau_n = \frac{h^2}{3} y'''(\xi_n), \qquad \xi_n \in (t_{n-2}, t_n). \tag{5.1.4b}$$

Problems

1. Using the method of undetermined coefficients, we can make some general observations regarding the coefficients of (5.1.2). Show, for example, that

$$\sum_{i=0}^{k} \alpha_i = 0 \tag{5.1.5a}$$

if (5.1.2) is exact when $y(t) = 1$. Additionally, if (5.1.2) is exact when $y(t) = t^q$, $q = 1, 2, \ldots$, show

$$\sum_{i=1}^{k} i^q \alpha_i + q \sum_{i=0}^{k} i^{q-1} \beta_i = 0, \qquad q = 1, 2, \ldots . \tag{5.1.5b}$$

These are order conditions for the linear multistep method.

5.2 Newton Divided-Difference Polynomials

Specific multistep formulas will be derived by approximating $y(t)$ or $f(t, y)$ by interpolating polynomials and, respectively, differentiating or integrating these polynomials. We've already seen examples where one-step methods were constructed by using interpolating polynomials with collocation. Thus, in this section, we'll review polynomial interpolation in an abstract setting that is removed from our from our primary task of solving differential equations.

The interpolation problem consists of finding a polynomial $P_k(t)$ of degree k that interpolates a function $f(t)$ at $k + 1$ distinct points $t_0, t_1, \ldots, t_k$, i.e.,

$$f(t_j) = P_k(t_j), \qquad j = 0, 1, \ldots, k. \tag{5.2.1}$$

In practice, we express the polynomial in a convenient basis for the application. The obvious basis of powers of monomial terms

$$P_k(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_k t^k = \sum_{i=0}^{k} a_i t^k \tag{5.2.2}$$

leads to the linear algebraic system

$$f(t_0) = a_0 + a_1 t_0 + a_2 t_0^2 + \ldots + a_k t_0^k,$$

$$f(t_1) = a_0 + a_1 t_1 + a_2 t_1^2 + \ldots + a_k t_1^k,$$

$$\ldots$$

$$f(t_k) = a_0 + a_1 t_k + a_2 t_k^2 + \ldots + a_k t_k^k,$$

and is rarely convenient. The Lagrange basis

$$L_i(t) = \prod_{j=1, j\neq i}^{k} \frac{t - t_j}{t_i - t_j} = \quad \frac{(t-t_0)(t-t_1)\dots(t-t_{i-1})(t-t_{i+1})\dots(t-t_k)}{(t_i-t_0)(t_i-t_1)\dots(t_i-t_{i-1})(t_i-t_{i+1})\dots(t_i-t_k)},$$
$$i = 0, 1, \dots, k, \tag{5.2.3a}$$

leads to the more direct solution

$$P_k(t) = \sum_{i=0}^{k} f(t_i) L_i(t). \tag{5.2.3b}$$

A disadvantage of the Lagrange basis is that the computation of the basis elements $L_i(t)$, $i = 0, 1, \dots, k$, must be repeated when changing the polynomial degree by adding or deleting an interpolation point. As discussed in Chapter 4, the Aitken-Neville recursion remedies this by obtaining $P_k(t)$ as a combination of polynomials of degree $k-1$. Thus, we define the family of interpolants

$$P_j^i(t) = (t - t_{i+j}) P_{j-1}^i(t) + (t_i - t) P_{j-1}^{i+1}(t) t_i - t_i + j, \qquad i = 0, 1, \dots, k-j,$$
$$j = 0, 1, \dots, k, \tag{5.2.4a}$$

where $P_j^i(t)$ is a polynomial of degree j satisfying the interpolation conditions

$$P_j^i(t_m) = f(t_m), \qquad m = i, i+1, \dots, i+j. \tag{5.2.4b}$$

The desired interpolating polynomial is $P_k^0(t)$. The Aitken-Neville algorithm simplifies degree changes, but its complexity is slightly larger than necessary ([6], Chapter 3). The Newton form of the interplating polynomial has a lower complexity and, like the Aitken-Neville procedure, is hierarchical. Again we'll define a sequence of polynomials of increasing degree with

$$P_0(t) = f(t_0). \tag{5.2.5}$$

The degree one polynomial is obtained by adding a correction to $P_0(t)$ in the form

$$P_1(t) = P_0(t) + a_1(t - t_0).$$

First, we easily verify that

$$P_1(t_0) = P_0(t_0) = f(t_0).$$

The coefficient a_1 is determined so that

$$P_1(t_1) = f(t_1);$$

thus, using (5.2.5),

$$f(t_1) = f(t_0) + a_1(t_1 - t_0)$$

and

$$a_1 = \frac{f(t_1) - f(t_0)}{t_1 - t_0}.$$

The notation is simplified by defining the *first divided difference* at the points t_j and t_l as

$$f[t_j, t_l] = \frac{f(t_j) - f(t_l)}{t_j - t_l}. \tag{5.2.6a}$$

Then

$$P_1(t) = P_0(t) + f[t_0, t_1](t - t_0). \tag{5.2.6b}$$

In a similar manner, the second-degree polynomial is written in the form

$$P_2(t) = P_1(t) + a_2(t - t_0)(t - t_1).$$

By construction, $P_2(t)$ satisfies the interpolation requirements at t_0 and t_1. Satisfaction of the interplation condition $P_2(t_2) = f(t_2)$ determines a_2 as

$$a_2 = f[t_0, t_1, t_2]$$

where $f[t_j, t_l, t_m]$ is the *second divided difference* at the points t_j, t_l, t_m

$$f[t_j, t_l, t_m] = \frac{f[t_j, t_l] - f[t_l, t_m]}{t_j - t_m}. \tag{5.2.7a}$$

Thus, the second divided difference is a divided difference of first divided differences. The second-degree interpolating polynomial is

$$P_2(t) = P_1(t) + f[t_0, t_1, t_2](t - t_0)(t - t_1). \tag{5.2.7b}$$

Continuing, we construct the k *th*-degree polynomial as

$$P_k(t) = P_{k-1}(t) + f[t_0, t_1, \ldots, t_k](t - t_0)(t - t_1)\ldots(t - t_k - 1) \tag{5.2.8a}$$

where

$$f[t_0, t_1, \ldots, t_k] = \frac{f[t_0, t_1, \ldots, t_{k-1}] - f[t_1, t_2, \ldots, t_k]}{t_0 - t_k} \tag{5.2.8b}$$

is the *k th divided difference* at the points t_0, t_1, ..., t_k.

The intermediate polynomials can be eliminated from (5.2.8a) to write the *Newton divided-difference polynomial* in the more explicit form

$$P_k(t) = \sum_{i=0}^{k} f[t_0, t_1, \ldots, t_i] \prod_{j=0}^{i-1} (t - t_j) \tag{5.2.9}$$

where the *zeroth divided difference* is $f[t_j] = f(t_j)$.

The Newton divided-difference representation is the traditional interpolating polynomial to be used when developing multistep formulas. It had some advantages for hand computation and the divided differences furnish approximations of solution derivatives that may, e.g., be used for error estimation. Before specializing the approximation (5.2.9) to our ODE application, let us note that the interpolation problem has a unique solution as indicated by the following theorem. The different bases just simplify the interpolation problem for specific applications.

Theorem 5.2.1. *There is one and only one polynomial of degree k that interpolates a function $f(t)$ at $k + 1$ distinct points.*

Proof. Suppose that there are two polynomials $P_k(t)$ and $Q_k(t)$ of degree k that interpolate $f(t)$ at the points $t_0, t_1, \ldots, t_k$. Subtract the two polynomials and define

$$R(t) = P_k(t) - Q_k(t).$$

Now $R(t)$ is also a polynomial of degree k that satisfies

$$R(t_j) = P_k(t_j) - Q_k(t_j) = 0, \qquad j = 0, 1, \ldots, k.$$

This, however, is impossible since a polynomial of degree k can only have k roots. Thus, the interpolating polynomial is unique. □

We may suspect that divided differences are related to derivatives and the following Lemma shows that this is the case.

Lemma 5.2.1. *Let $f(t) \in C^k[a,b]$ and let $t_0, t_1, \ldots, t_k$ be $k+1$ distinct points on $[a,b]$, then*

$$f[t_0, t_1, \ldots, t_k] = \frac{f^{(k)}(\xi)}{k!}, \qquad \xi \in (a,b). \tag{5.2.10}$$

Proof. Consider the function

$$g(t) = f(t) - P_k(t).$$

Since $f(t_j) = P_k(t_j)$, $j = 0, 1, \ldots, k$, the function $g(t)$ has $k+1$ distinct zeros in $[a,b]$. Thus, according to Rolle's theorem, $g'(t)$ vanishes at k distinct points on (a,b) (Figure 5.2.1). Similarly, $g''(t)$ vanishes at $k-1$ points on (a,b) and, continuing in this manner, $g^{(k)}$ will vanish at one point on (a,b). Let us call this point ξ. Thus,

$$g^{(k)}(\xi) = f^{(k)}(\xi) - P_k^{(k)}(\xi) = 0$$

According to (5.2.8a) or (5.2.9), the k *th* derivative of $P_k(t)$ is

$$P_k^{(k)}(t) = k! f[t_0, t_1, \ldots, t_k]$$

Combining the above two results establishes the result. □

With Lemma 5.2.1, we can obtain formulas for interpolation errors.

Theorem 5.2.2. *Let $f(t) \in C^{k+1}[a,b]$ and let $t_0, t_1, \ldots, t_k$ be $k+1$ distinct points on $[a,b]$, then there exists a point $\xi = \xi(t) \in (a,b)$ such that*

$$E_k(t) = f(t) - P_k(t) = \frac{f^{(k+1)}(\xi)}{(k+1)!} \prod_{j=0}^{k} (t - t_j). \tag{5.2.11}$$

Proof. Construct a polynomial $P_{k+1}(t)$ that interpolates $f(t)$ at $t_0, t_1, \ldots, t_k$ and the additional point τ. According to (5.2.8a),

$$P_{k+1}(t) = P_k(t) + f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (t - t_j).$$

Since $P_{k+1}(\tau) = f(\tau)$, we have

$$f(\tau) = P_k(\tau) + f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (\tau - t_j).$$

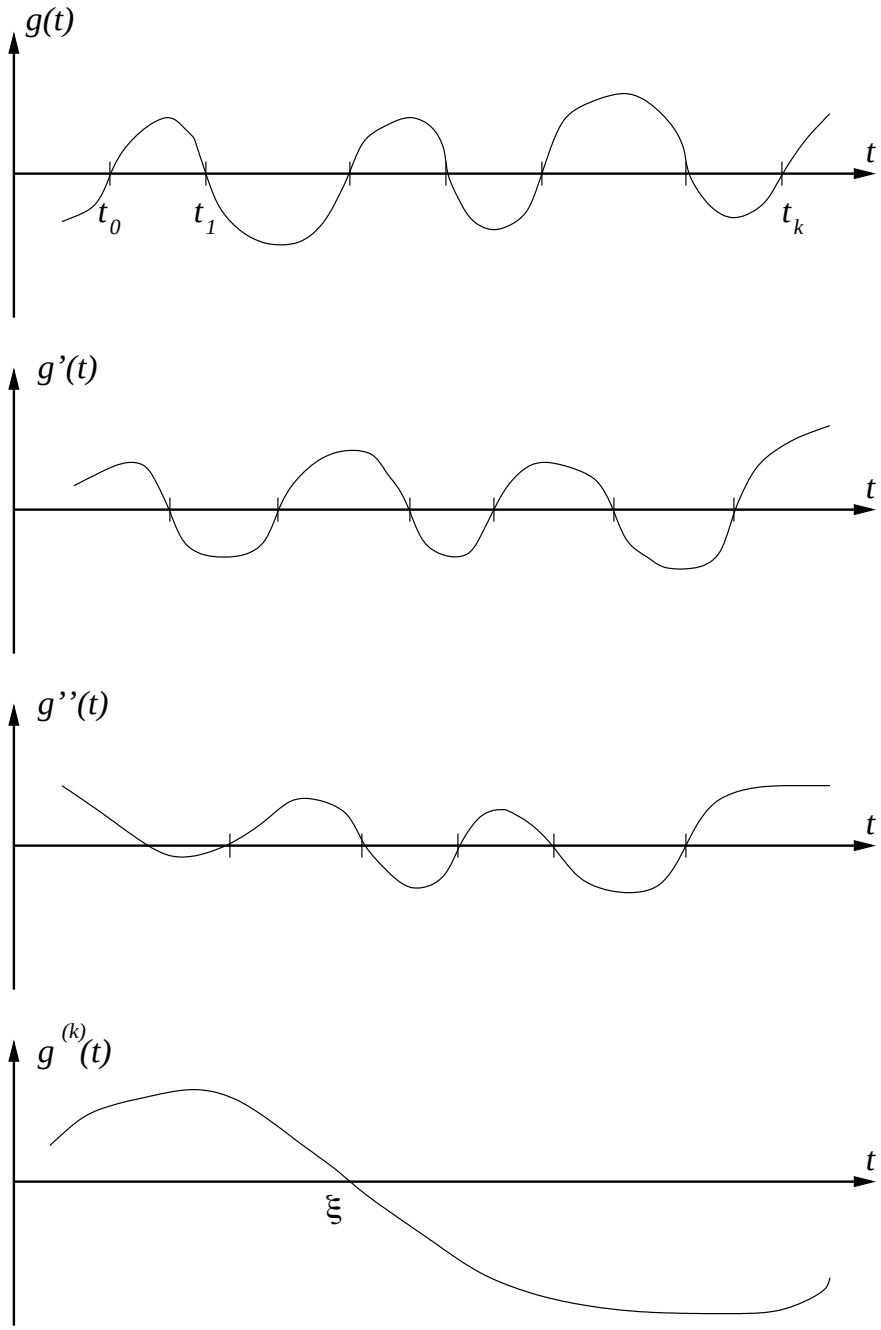

Figure 5.2.1: Zeros and extrema of the function $g(t) = f(t) - P_k(t)$.

Thus,

$$E_k(\tau) = f(\tau) - P_k(\tau) = f[t_0, t_1, \ldots, t_k, \tau] \prod_{j=0}^{k} (\tau - t_j).$$

Using (5.2.10)

$$E_k(\tau) = \frac{f^{(k+1)}(\xi)}{(k+1)!} \prod_{j=0}^{k} (\tau - t_j).$$

Since τ is arbitrary we have established the result. □

The points in the divided-difference polynomial have to be distinct but do not have to be uniform or ordered. However, the computation simplifies greatly when the interpolation points are ordered and uniformly spaced. This will be our main interest when developing formulas for ODEs, so we'll restrict

$$t_i = t_0 + ih, \qquad i = 0, 1, \ldots, k. \tag{5.2.12}$$

With this, let us define the *first forward* and *backward difference operators* as

$$\Delta f_i = f_{i+1} - f_i, \tag{5.2.13a}$$

$$\nabla f_i = f_i - f_{i-1}. \tag{5.2.13b}$$

Although both operators are used, backward differences better suit our current needs (because multistep methods will be interpolating with data at prior times). Thus, using (5.2.6a) and (5.2.13b), let us write the first divided difference with uniform spacing as

$$f[t_{i-1}, t_i] = \frac{f_{i-1} - f_i}{t_{i-1} - t_i} = \frac{\nabla f_i}{h}. \tag{5.2.14}$$

Higher-order operators follow by iteration; thus, using (5.2.7a)

$$f[t_{i-2}, t_{i-1}, t_i] = \frac{f[t_{i-2}, t_{i-1}] - f[t_{i-1}, t_i]}{t_{i-2} - t_i} = \frac{\nabla f_i - \nabla f_{i-1}}{2h^2} = \frac{f_i - 2f_{i-1} + f_{i-2}}{2h^2}.$$

In a similar manner, we define the higher-order backward differences recursively as

$$\nabla^m f_i = \nabla^{m-1} f_i - \nabla^{m-1} f_i - 1 \tag{5.2.15a}$$

with the understanding that

$$\nabla^0 f_i = f_i. \tag{5.2.15b}$$

Thus,

$$f[t_{i-2}, t_{i-1}, t_i] = \frac{\nabla^2 f_i}{2h^2}. \tag{5.2.15c}$$

The $k\ th$ divided difference adn $k\ th$ backward difference operator are related by (Problem 1)

$$f[t_{i-k}, t_{i-k+1}, \ldots, t_i] = \frac{\nabla^k f_i}{k!h^k}. \tag{5.2.15d}$$

We can easily write the divided-difference polynomial (5.2.9) in terms of backward differences; however, our intended use with ODEs will call for polynomials proceeding from an advanced point (t_{n-1} or t_n) into the past. Hence, it will be convenient to re-index the points in (5.2.9) to account for this. For the present, We'll illustrate this by reversing the indexing; thus, let t_0 become t_k, t_1 become t_{k-1} etc. Then, (5.2.9) becomes

$$P_k(t) = \sum_{i=0}^{k} f[t_k, t_{k-1}, \ldots, t_{k-i}] \prod_{j-1}^{i-1} (t - t_{k-j})$$

or

$$P_k(t) = f[t_k] + f[t_k, t_{k-1}](t - t_k) + \ldots + f[t_k, t_{k-1}, \ldots, t_0](t - t_k)(t - t_{k-1}) \ldots (t - t_1).$$

Now, for uniform spacing, we use (5.2.15) to obtain

$$P_k(t) = \sum_{i=0}^{k} \frac{\nabla^i f_k}{i!h^i} \prod_{j=0}^{i-1} (t - t_{k-j}) \tag{5.2.16a}$$

or

$$P_k(t) = f_k + \frac{\nabla f_k}{h}(t - t_k) + \ldots + \frac{\nabla^k f_k}{k!h^k}(t - t_k)(t - t_{k-1}) \ldots (t - t_1). \tag{5.2.16b}$$

Since our primary purpose is the development of multistep methods for ODEs, we won't illustrate any interpolation examples at this point but will illustrate use of (5.2.16) in subsequent sections.

Problems

1. *Problem 1.* Derive the identity (5.2.15d) using, e.g., an induction argument.

5.3 Explicit Methods: Adams-Bashforth Methods

The first multistep formulas that we consider are explicit methods called Adams-Bashforth methods. These are derived by integrating the ODE

$$y' = f(t, y) \tag{5.3.1}$$

on the interval (t_{n-1}, t_n) to obtain

$$\int_{t_{n-1}}^{t_n} y' dt = \int_{t_{n-1}}^{t_n} f(t, y(t)) dt$$

or

$$y(t_n) = y(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(t, y(t)) dt. \tag{5.3.2}$$

Specific numerical techniques are obtained by approximating $f(t, y(t))$ in (5.3.2) by an interpolating polynomial and integrating the result. If, for example, $f(t, y(t))$ were approximated by (the constant interpolating polynomial) $f(t, y(t_{n-1}))$ then (5.3.2) would produce Euler's method

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}).$$

More generally, we'll interpolate $f(t, y(t))$ by a $k-1$ *st* degree polynomial passing through t_{n-1}, t_{n-2}, ..., t_{n-k}. We'll use the Newton backward-difference form of the interplating polynomial. For this application, we identify interpolation point t_k in (5.2.16) with t_{n-1}, t_{k-1} with t_{n-2}, ..., and t_1 with t_{n-k}. Then, (5.2.16) becomes

$$P_{k-1}(t) = \sum_{i=0}^{k-1} \frac{\nabla^i f_{n-1}}{i! h^i} \prod_{j=0}^{i-1} (t - t_{n-1-j}). \tag{5.3.3a}$$

Using (5.2.11) with k replaced by $k-1$ and re-indexing the interpolation points as described above, we determine the error of this interpolation as

$$E_{k-1}(t) = f(t, y(t)) - P_{k-1}(t) = \frac{f^{(k)}(\xi, y(\xi))}{k!} \prod_{j=0}^{k-1} (t - t_{n-1-j}), \qquad \xi \in (t_{n-k}, t_{n-1}). \tag{5.3.3b}$$

Let's expand both (5.3.3a) and (5.3.3b) to reveal their structure

$$P_{k-1}(t) = f_{n-1} + \frac{\nabla f_{n-1}}{h}(t - t_{n-1}) + \frac{\nabla^2 f_{n-1}}{2! h^2}(t - t_{n-1})(t - t_{n-2})$$

$$+\ldots+\frac{\nabla^{k-1}f_{n-1}}{(k-1)!h^{k-1}}(t-t_{n-1})(t-t_{n-2})\ldots(t-t_{n-k+1})$$

and

$$E_{k-1}(t)=\frac{f^{(k)}(\xi,y(\xi))}{k!}(t-t_{n-1})(t-t_{n-2})\ldots(t-t_{n-k}).$$

Using (5.2.14, 5.2.15), the first few divided differences are

$$\nabla^0 f_{n-1}=f_{n-1},\qquad \nabla f_{n-1}=f_{n-1}-f_{n-2},$$

$$\nabla^2 f_{n-1}=\nabla f_{n-1}-\nabla f_{n-2}=f_{n-1}-2f_{n-2}+f_{n-3}.$$

The integration (5.3.2) can be simplified by the change of variables

$$\tau=\frac{t-t_{n-1}}{h} \tag{5.3.4a}$$

Since $t_{n-1-j}=t_{n-1}-jh$, we have

$$t-t_{n-1-j}=(\tau+j)h$$

and

$$\frac{1}{h^i}\prod_{j=0}^{i-1}(t-t_{n-1-j})=\prod_{j=0}^{i-1}(\tau+j)=\tau(\tau+1)\ldots(\tau+i-1).$$

Recall the combination symbol

$$\left(\begin{array}{c}\tau\\ i\end{array}\right)=\frac{\tau!}{i!(\tau-i)!}=\frac{\tau(\tau-1)\ldots(\tau-i+1)}{i!},\qquad \left(\begin{array}{c}\tau\\ 0\end{array}\right)=1. \tag{5.3.4b}$$

This formula makes sense when τ is negative, i.e.,

$$\left(\begin{array}{c}-\tau\\ i\end{array}\right)=\frac{-\tau(-\tau-1)\ldots(-\tau-i+1)}{i!}=(-1)^i\frac{\tau(\tau+1)\ldots(\tau+i-1)}{i!}.$$

Thus,

$$\frac{1}{i!h^i}\prod_{j=0}^{i-1}(t-t_{n-1-j})=(-1)^i\left(\begin{array}{c}-\tau\\ i\end{array}\right). \tag{5.3.4c}$$

Substituting (5.3.4c) into (5.3.3a,b) yields

$$P_{k-1}(t)=\sum_{i=0}^{k-1}(-1)^i\left(\begin{array}{c}-\tau\\ i\end{array}\right)\nabla^i f_{n-1} \tag{5.3.5a}$$

and, using (5.3.1),

$$E_{k-1}(t) = (-1)^k h^k \binom{-\tau}{k} y^{(k+1)}(\xi). \tag{5.3.5b}$$

Replacing $f(t, y(t))$ in (5.3.2) by $P_{k-1}(t) + E_{k-1}(t)$ and transforming the integrals from t to τ using (5.3.4a) yields

$$y(t_n) = y(t_{n-1}) + h \int_0^1 \sum_{i=0}^{k-1} (-1)^i \binom{-\tau}{i} \nabla^i f_{n-1} d\tau + h \int_0^1 (-1)^k h^k \binom{-\tau}{k} y^{(k+1)}(\xi) d\tau.$$

Let

$$\gamma_m = (-1)^m \int_0^1 \binom{-\tau}{m} d\tau. \tag{5.3.6a}$$

Then we have

$$y(t_n) = y(t_{n-1}) + h \sum_{i=0}^{k-1} \gamma_i \nabla^i f_{n-1} + (-1)^k h^{k+1} \int_0^1 \binom{-\tau}{k} y^{(k+1)}(\xi) d\tau. \tag{5.3.6b}$$

Dropping the error term yields the *k-step Adams-Bashforth* method

$$y_n = y_{n-1} + h \sum_{i=0}^{k-1} \gamma_i \nabla^i f_{n-1}. \tag{5.3.7a}$$

The local error of this formula is

$$d_n = (-1)^k h^{k+1} \int_0^1 \binom{-\tau}{k} y^{(k+1)}(\xi) d\tau. \tag{5.3.7b}$$

Comparing (5.3.7a) with the general multistep formula (5.1.2a), we see that $\alpha_0 = 1$, $\alpha_1 = -1$, and $\alpha_2 = \alpha_3 = \ldots = \alpha_k = \beta_0 = 0$. Thus, (5.3.7a) uses the one solution value y_{n-1} and the k function values f_{n-1}, f_{n-2}, $\ldots$, f_{n-k} to obtain the y_n (cf. Figure 5.3.1). We'll call (5.3.7a) a k-step method since it uses solution and function values at the k points t_{n-1}, t_{n-2}, $\ldots$, t_{n-k}; however, other terminology is also used [10].

From (5.3.7b) we see that the local error of (5.3.7a) is $O(h^{k+1})$. Thus, the order of a k-step Adams-Bashforth method is k. The method (5.3.7) has only one function evaluation per step. This is contrasted to the minimum of k function evaluations that are needed for a k-stage Runge-Kutta method.

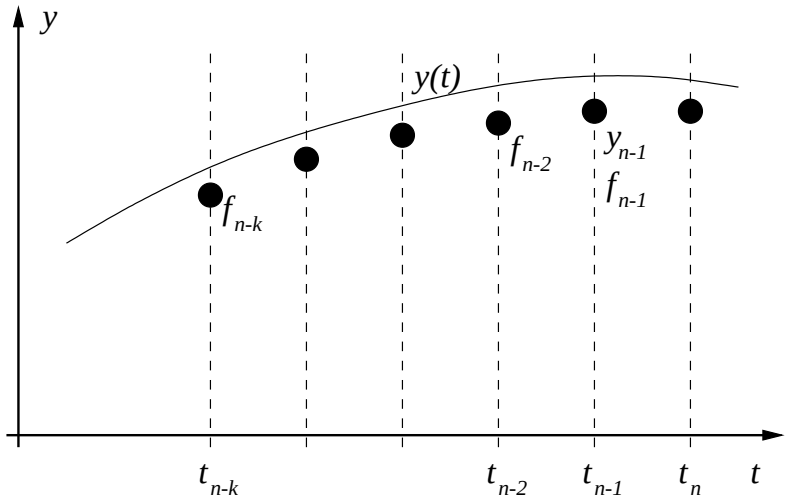

Figure 5.3.1: Information needed for a k-step Adams-Bashforth method.

The representation of (5.3.7a) in terms of backward differences will be useful to us in some circumstances, but generally we prefer a formula written in terms of function values at preceding points. This can be done using

$$\nabla^q f_{n-1} = \sum_{j=0}^{q} (-1)^j \begin{pmatrix} q \\ j \end{pmatrix} f_{n-1-j}. \tag{5.3.8a}$$

Then, using (5.3.7a)

$$y_n = y_{n-1} + h \sum_{i=0}^{k-1} \gamma_i \sum_{j=0}^{i} (-1)^j \begin{pmatrix} i \\ j \end{pmatrix} f_{n-1-j}. \tag{5.3.8b}$$

By rearranging the order of the computation (5.3.8b) can be written in the form (Problem 1)

$$y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j f_{n-j} \tag{5.3.8c}$$

where

$$\beta_j = \sum_{i=j-1}^{k-1} \gamma_i \begin{pmatrix} i \\ j-1 \end{pmatrix}. \tag{5.3.8d}$$

The formula (5.3.8d) now has the form of (5.1.2a).

The parameters γ_j of (5.3.6a) and β_j of (5.3.8d) are independent of the problem and can be evaluated in advance of the computation. Thus, using (5.3.6a)

$$\gamma_0 = \int_0^1 \begin{pmatrix} -\tau \\ 0 \end{pmatrix} d\tau = 1,$$

$$\gamma_1 = \int_0^1 -\begin{pmatrix} -\tau \\ 1 \end{pmatrix} d\tau = \int_0^1 \tau d\tau = \frac{1}{2},$$

$$\gamma_2 = \int_0^1 \begin{pmatrix} -\tau \\ 2 \end{pmatrix} d\tau = \int_0^1 \frac{-\tau(-\tau-1)}{2} d\tau = \frac{5}{12}.$$

Continuing in this manner and using the results in (5.3.7a) yields

$$y_n = y_{n-1} + h(1 + \frac{1}{2}\nabla + \frac{5}{12}\nabla^2 + \frac{3}{8}\nabla^3 + \frac{251}{720}\nabla^4 + \frac{95}{288}\nabla^5 + \ldots + \gamma_{k-1}\nabla^{k-1})f_{n-1}. \tag{5.3.9a}$$

Additional results, given in Hairer *et al.* [12], Section III.1, are reproduced in Table 5.3.1.

j	0	1	2	3	4	5	6	7	8
γ_j	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$	$\frac{5257}{17280}$	$\frac{1070017}{3628800}$

Table 5.3.1: Coefficients γ_j for Adams-Bashforth methods ([12], Section III.1).

You may recall the second mean value theorem for integrals which states that if $p(\tau) \in C^0[a,b]$ and $q(\tau)$ is integrable and does not change sign on $[a,b]$ then

$$\int_a^b p(\tau)q(\tau)d\tau = p(\eta)\int_a^b q(\tau)d\tau, \qquad \eta \in (a,b).$$

Examining (5.3.4b), reveals that

$$\begin{pmatrix} -\tau \\ k \end{pmatrix} = \frac{\tau(\tau+1)\ldots(\tau+k-1)}{k!}$$

does not change sign for $\tau \in [0,1]$. Thus, we can use the second mean value theorem with (5.3.7b) and write the local error in the more explicit form

$$d_n = (-1)^k h^{k+1} y^{(k+1)}(\eta) \int_0^1 \begin{pmatrix} -\tau \\ k \end{pmatrix} d\tau = \gamma_k h^{k+1} y^{(k+1)}(\eta), \qquad \eta \in (t_{n-1}, t_n). \tag{5.3.9b}$$

Thus, the error coefficient γ_k is also known from, e.g., Table 5.3.1. Contrast this with discretization error formulas for Runge-Kutta methods that were extremely complex.

Formulas and their local discretization errors for $k = 1, 2, 3, 4$, follow.

- $k = 1$: Euler's method

$$y_n = y_{n-1} + hf_{n-1}, \tag{5.3.10a}$$

$$\tau_n = \frac{h}{2}y''(\xi), \tag{5.3.10b}$$

- $k = 2$: Two-step Adams-Bashforth formula

$$y_n = y_{n-1} + h(f_{n-1} + \nabla f_{n-1}/2)$$

or

$$y_n = y_{n-1} + \frac{h}{2}(3f_{n-1} - f_{n-2}), \tag{5.3.11a}$$

$$\tau_n = \frac{5}{12}h^2 y'''(\xi). \tag{5.3.11b}$$

- $k = 3$: Three-step Adams-Bashforth formula

$$y_n = y_{n-1} + \frac{h}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3}), \tag{5.3.12a}$$

$$\tau_n = \frac{3}{8}h^3 y^{iv}(\xi). \tag{5.3.12b}$$

- $k = 4$: Four-step Adams-Bashforth formula

$$y_n = y_{n-1} + \frac{h}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}), \tag{5.3.13a}$$

$$\tau_n = \frac{251}{720}h^4 y^{v}(\xi). \tag{5.3.13b}$$

The coefficients of the Adams-Bashforth methods (5.3.8c) and the local error coefficient according to (5.3.9b) are repeated in Table 5.3.2 for $k = 1, 2, \ldots, 6$ [1].

k		$j=1$	2	3	4	5	6	γ_k
1	β_j	1						$\frac{1}{2}$
2	$2\beta_j$	3	-1					$\frac{5}{12}$
3	$12\beta_j$	23	-16	5				$\frac{3}{8}$
4	$24\beta_j$	55	-59	37	-9			$\frac{251}{720}$
5	$720\beta_j$	1901	-2774	2616	-1274	251		$\frac{95}{288}$
6	$1440\beta_j$	4277	-7923	9982	-7298	2877	-475	$\frac{19087}{60480}$

Table 5.3.2: Coefficients of the Adamas-Bashforth method (5.3.8c) and the local error (5.3.9b) for orders one through six [1].

Problems

1. Show that (5.3.8b) can be written in the form

$$y_n = y_{n-1} + h\sum_{j=0}^{k-1}(-1)^j f_{n-1-j}\sum_{i=j}^{k-1}\gamma_i \begin{pmatrix} i \\ j \end{pmatrix}$$

and that this leads to (5.3.8c).

5.4 Implicit Methods: Adams-Moulton Methods

Implicit Adams formulas, called Adams-Moulton methods, are derived in the same manner as the explicit formulas of Section 5.3. In this case, we approximate f by a $k-1$ st degree interpolating polynomial $P_{k-1}(t)$ that satisfies

$$P_{k-1}(t_{n-i}) = f(t_{n-i}, y(t_{n-i})), \qquad i = 0, 1, \ldots, k-1. \tag{5.4.1}$$

A formula for $P_{k-1}(t)$ and its error follow from (5.3.3a) and (5.3.3b) upon replacement of $n-1$ by n. This yields

$$P_{k-1}(t) = \sum_{i=0}^{k-1}\frac{\nabla^i f_n}{i!h^i}\prod_{j=0}^{i-1}(t - t_{n-j}) \tag{5.4.2a}$$

$$E_{k-1}(t) = \frac{f^(k)(\xi, y(\xi))}{k!}\prod_{j=0}^{k-1}(t - t_{n-j}). \tag{5.4.2b}$$

Again letting $\tau = (t - t_{n-1})/h$ and substituting (5.4.2) into (5.3.2), we find

$$y(t_n) = y(t_{n-1}) + h\int_0^1 \sum_{i=0}^{k-1}(-1)^i \begin{pmatrix} -\tau+1 \\ i \end{pmatrix}\nabla^i f_n d\tau +$$

$$h\int_0^1 (-1)^k h^k \begin{pmatrix} -\tau+1 \\ k \end{pmatrix} y^{(k+1)}(\xi)d\tau.$$

Letting

$$\gamma_i^* = \int_0^1 (-1)^i \begin{pmatrix} -\tau+1 \\ i \end{pmatrix} d\tau. \tag{5.4.3a}$$

we find the Adams-Moulton formula as

$$y_n = y_{n-1} + h\sum_{i=0}^{k-1}\gamma_i^*\nabla^i f_n. \tag{5.4.3b}$$

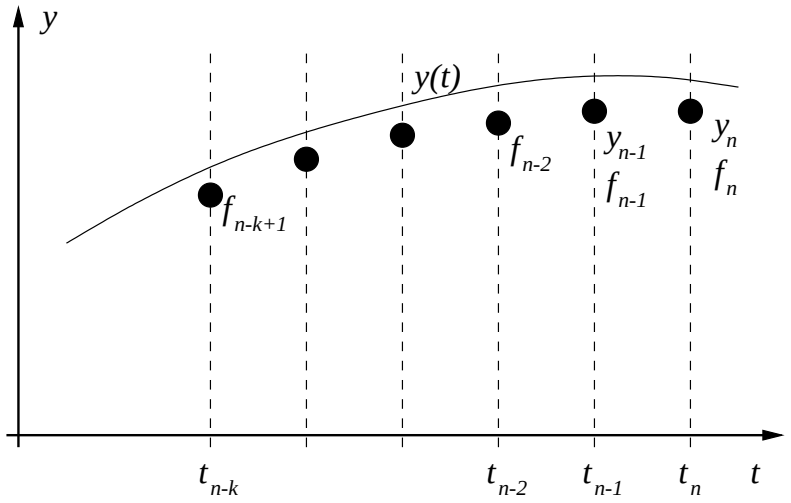

Figure 5.4.1: Information needed for a k-value Adams-Moulton method.

The local error is, as usual, a bit more complex with implicit formulas. We'll skirt this by using the local discretization error

$$\tau_n = \frac{y(t_n) - y(t_{n-1})}{h} - \sum_{i=0}^{k-1} \gamma_i^* \nabla^i f_n$$

or

$$\tau = (-1)^k h^k \int_0^1 \binom{-\tau+1}{k} y^{(k+1)}(\xi) d\tau = \gamma_k^* h^k y^{(k+1)}(\eta), \qquad \eta \in (t_{n-1}, t_n). \quad (5.4.3c)$$

As shown in Figure 5.4.1, the solution y_{n-1} and function values f_n, f_{n-1}, ..., f_{n-k+1} are needed to compute y_n. Unlike the Adams-Bashforth formula, the index k does not indicate the number of steps of the method. Methods with $k = 1$ and 2 are one-step methods and those with $k \geq 3$ are $k-1$-step methods. We might call the k *th* Adams-Moulton method a k-value method since it involves k values of f. Like a k-step Adams-Bashforth methods, a k-value Adams-Moulton method is of order k (cf. (5.4.3c)).

For linear problems, (5.4.3a,b) only requires one function evaluation per step; however, iteration will generally be needed for nonlinear problems.

Expanding (5.4.3b) by evaluating γ_i^*, $i = 0, 1, \ldots, k-1$, according to (5.4.3a), we find

$$y_n = y_{n-1} + h(1 - \frac{1}{2}\nabla - \frac{1}{12}\nabla^2 - \frac{1}{24}\nabla^3 - \frac{19}{720}\nabla^4 - \frac{3}{160}\nabla^5 +$$

$$\ldots + \gamma_{k-1}^* \nabla^k - 1) f_n. \tag{5.4.4}$$

Additional results, given in Hairer *et al.* [12], Section III.1, are reproduced in Table 5.4.1.

j	0	1	2	3	4	5	6	7	8
γ_j^*	1	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$	$-\frac{275}{24192}$	$-\frac{33953}{3628800}$

Table 5.4.1: Coefficients γ_j^* for implicit Adams methods ([12], Section III.1).

Expanding the backward differences in (5.4.4) using (5.2.15a,b), we find specific formulas for given choices of k. Those formulas for $k = 1, 2, 3, 4$, and their local discretization errors follow.

- $k = 1$: Backward Euler method

$$y_n = y_{n-1} + h f_n, \tag{5.4.5a}$$

$$\tau_n = -\frac{h}{2} y''(\xi). \tag{5.4.5b}$$

- $k = 2$: Trapezoidal rule

$$y_n = y_{n-1} + h(f_n - \frac{1}{2}\nabla f_n)$$

or

$$y_n = y_{n-1} + \frac{h}{2}(f_n + f_{n-1}), \tag{5.4.6a}$$

$$\tau_n = -\frac{1}{12} h^2 y'''(\xi). \tag{5.4.6b}$$

- $k = 3$: Three-value (two-step) Adams-Moulton formula

$$y_n = y_{n-1} + \frac{h}{12}(5 f_n + 8 f_{n-1} + f_{n-2}), \tag{5.4.7a}$$

$$\tau_n = -\frac{1}{24} h^3 y^{iv}(\xi). \tag{5.4.7b}$$

- $k = 4$: Four-value (three-step) Adams-Moulton formula

$$y_n = y_{n-1} + \frac{h}{24}(9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3}), \tag{5.4.8a}$$

$$\tau_n = -\frac{19}{720}h^4 y^v(\xi). \tag{5.4.8b}$$

Once again, we present the coefficients of the Adams-Moulton methods

$$y_n = y_{n-1} + h\sum_{j=0}^{k-1} \beta_j f_{n-j} \tag{5.4.9}$$

and their error coefficients (5.4.3c) for $k = 1, 2, \ldots, 6$, in Table 5.4.2.

k		$j=0$	1	2	3	4	5	γ_k
1	β_j	1						$-\frac{1}{2}$
2	$2\beta_j$	1	1					$-\frac{1}{12}$
3	$12\beta_j$	5	8	-1				$-\frac{1}{24}$
4	$24\beta_j$	9	19	-5	1			$-\frac{19}{720}$
5	$720\beta_j$	251	646	-264	106	-19		$-\frac{3}{160}$
6	$1440\beta_j$	475	1427	-798	482	-173	27	$-\frac{863}{60480}$

Table 5.4.2: Coefficients of the Adams-Moulton method (5.4.9) and their local error coefficients (5.4.3c) for orders one through six [1].

Example 5.4.1. Comparing Tables 5.3.2 and 5.4.2, we see that the error coefficient of the Adams-Moulton method of order k is smaller than that of the Adams-Bashforth method of the same order. Thus, with comparable derivatives, the implicit methods should produce more accuracy than the explicit methods. Let's examine this possibility using the simple example (cf. Burden and Faires [6], Chapter 5)

$$y' = y - t^2 + 1, \qquad 0 < t \le 2, \qquad y(0) = \frac{1}{2},$$

which has the exact solution

$$y(t) = (t+1)^2 - \frac{e^t}{2}.$$

Burden and Faires [6] calculate solutions using the fourth order Adams-Bashforth (5.3.13a) and Adams-Moulton (5.4.8a) formulas. Since this problem is linear, the Adams-Moulton solution may be obtained without iteration. Results with $h = 0.2$ are shown in

Table 5.4.3. The four starting values (y_0, y_1, y_2, and y_3) that are needed for (5.3.13a) and the three (y_0, y_1, and y_2) that are needed for (5.4.8) are obtained from the exact solution. The global errors found when using the classical fourth-order Runge-Kutta method (3.2.4) with the same step size are also shown in Table 5.4.3.

t_n	$y(t_n)$	Adams-Bashforth Error	Adams-Moulton Error	Runge-Kutta Error
0.0	0.5000000	0.0000000	0.0000000	0.0000000
0.2	0.8292986	0.0000000	0.0000000	0.0000053
0.4	1.2140877	0.0000000	0.0000000	0.0000114
0.6	1.6489406	0.0000000	0.0000065	0.0000186
0.8	2.1272207	0.0000828	0.0000160	0.0000269
1.0	2.6408227	0.0002219	0.0000293	0.0000364
1.2	3.1798942	0.0004065	0.0000478	0.0000474
1.4	3.7323401	0.0006601	0.0000731	0.0000599
1.6	4.2834095	0.0010093	0.0001071	0.0000743
1.8	4.8150857	0.0014812	0.0001527	0.0000906
2.0	5.3053630	0.0021119	0.0002132	0.0001089

Table 5.4.3: Global errors for Example 5.4.1 obtained by fourth-order Adams-Bashforth, Adams-Moulton, and classical Runge-Kutta methods ([6], Chapter 5).

Errors with the Adams-Moulton method are about ten times smaller than those with the Adams-Bashforth method. The ratio of their error coefficients is $251/19 \approx 13$ (cf. (5.3.13b) and (5.4.8b)). Thus, the results are close to their expected values. Accuracy of the Runge-Kutta method is almost twice that of the Adams-Moulton method; however, the Runge-Kutta method requires approximately four times the work of the two Adams methods.

5.5 Implicit Methods: Backward-difference Methods

While we haven't analyzed stability of multistep methods, our experience with Runge-Kutta methods indicates that implicit methods have better stability properties than explicit methods. Thus, with the objectives of developing implicit methods having good stability properties for stiff systems and of illustrating another class of methods, we'll

examine *backward-difference formulas*. These methods are derived by approximating $y(t)$ by a k *th*-degree interpolating polynomial $P_k(t)$ using the $k+1$ points t_{n-i}, $i = 0, 1, \ldots, k$, and differentiating $P_k'(t)$ to approximate $y'(t)$. Once again, we'll use the Newton form of the interpolating polynomial which may be obtained from (5.3.3) by replacing $n-1$ by n, f by y, and $k-1$ by k to obtain

$$P_k(t) = \sum_{i=0}^{k} \frac{\nabla^i y(t_n)}{i! h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) \tag{5.5.1a}$$

$$E_k(t) = y(t) - P_k(t) = \frac{y^{(k+1)}(\xi)}{(k+1)!} \prod_{j=0}^{k} (t - t_{n-j}), \qquad \xi \in (t_{n-k}, t_n). \tag{5.5.1b}$$

The backward-difference method will be obtained by collocating at $t = t_n$, i.e., by enforcing

$$P_k'(t_n) = f(t_n, P_k(t_n)). \tag{5.5.2}$$

Once again, it will be convenient to change variables by letting

$$\tau = \frac{t - t_n}{h} \tag{5.5.3a}$$

and use (5.3.4) to obtain

$$\frac{1}{i! h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) = \frac{\tau(\tau+1)\ldots(\tau+i-1)}{i!} = (-1)^i \begin{pmatrix} -\tau \\ i \end{pmatrix}.$$

Then, (5.5.1) can be written as

$$P_k(t) = \sum_{i=0}^{k} (-1)^i \begin{pmatrix} -\tau \\ i \end{pmatrix} \nabla^i y(t_n) \tag{5.5.3b}$$

and

$$E_k(t) = (-1)^{k+1} h^{k+1} \begin{pmatrix} -\tau \\ k+1 \end{pmatrix} y^{(k+1)}(\xi). \tag{5.5.3c}$$

Differentiating (5.5.3b)

$$P_k'(t_n) = \frac{1}{h} \frac{dP_k(0)}{d\tau} = \frac{1}{h} \sum_{i=1}^{k} \delta_i \nabla^i y(t_n) \tag{5.5.4a}$$

where

$$\delta_i = (-1)^i \frac{d}{d\tau} \left(\begin{array}{c} -\tau \\ i \end{array} \right)_{\tau=0} = \frac{d}{d\tau} \left[\frac{\tau(\tau+1)(\tau+2)\ldots(\tau+i-1)}{i!} \right]_{\tau=0}, \qquad i \geq 1.$$

Differentiating the product yields

$$\delta_i = \frac{1}{i}, \qquad i \geq 1. \tag{5.5.4b}$$

Differentiation of the error expression (5.5.3b) proceeds in the same manner. Bear in mind that ξ is a function of t (or τ); however,

$$\left(\begin{array}{c} -\tau \\ k+1 \end{array} \right)_{\tau=0} = 0,$$

so

$$E_k'(t_n) = h^k \delta_{k+1} y^{(k+1)}(\xi(t_n)) = \frac{h^k}{k+1} y^{(k+1)}(\xi(t_n)). \tag{5.5.4c}$$

Replacing $y(t_n)$ in (5.3.1) by $P_k(t_n) + e_k(t_n)$ and using (5.5.4) yields

$$\sum_{i=1}^{k} \frac{\nabla^i y(t_n)}{i} + \frac{h^{k+1}}{k+1} y^{(k+1)}(\xi) = h f(t_n, y(t_n)).$$

Neglecting the local discretization error yields the backward-difference formula

$$\sum_{i=1}^{k} \frac{\nabla^i y_n}{i} = h f(t_n, y_n). \tag{5.5.5a}$$

Its local discretization error is

$$\tau_n = \sum_{i=1}^{k} \frac{\nabla^i y(t_n)}{i} - h f(t_n, y(t_n)) = -\frac{h^k}{k+1} y^{(k+1)}(\xi), \qquad \xi \in (t_{n-k}, t_n). \tag{5.5.5b}$$

The formula (5.5.5a) has the general form of (5.1.2a) with $\beta_1 = \beta_2 = \ldots = \beta_k = 0$, i.e.,

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = h \beta_0 f_n, \tag{5.5.5c}$$

The backward-difference formula with index k is a k-step method having order k (since the local discretization error is $O(h^k)$.

Backward difference methods and their local discretization errors for $k = 1, 2, 3, 4$, follow. Backward difference coefficients for $k = 1, 2, \ldots, 6$, appear in Table 5.5.1 [1].

- $k = 1$: Backward Euler method

$$y_n = y_{n-1} + hf_n, \tag{5.5.6a}$$

$$\tau_n = -\frac{h}{2}y''(\xi). \tag{5.5.6b}$$

- $k = 2$: Two-step backward-difference formula

$$\nabla y_n + \frac{\nabla^2 y_n}{2} = hf_n$$

or

$$y_n = \frac{1}{3}(4y_{n-1} - y_{n-2} + 2hf_n), \tag{5.5.7a}$$

$$\tau_n = -\frac{h^2}{3}y'''(\xi). \tag{5.5.7b}$$

- $k = 3$: Three-step backward-difference formula

$$y_n = \frac{1}{11}(18y_{n-1} - 9y_{n-2} + 2y_{n-3} + 6hf_n), \tag{5.5.8a}$$

$$\tau_n = -\frac{h^3}{4}y^{iv}(\xi). \tag{5.5.8b}$$

- $k = 4$: Four-step backward-difference formula

$$y_n = \frac{1}{25}(48y_{n-1} - 36y_{n-2} + 16y_{n-3} - 3y_{n-4} + 12hf_n), \tag{5.5.9a}$$

$$\tau_n = -\frac{h^4}{5}y^{v}(\xi). \tag{5.5.9b}$$

Examples using backward-difference formulas will be presented in Section 5.7.

k	δ	$\delta\beta_0$	$\delta\alpha_0$	$\delta\alpha_1$	$\delta\alpha_2$	$\delta\alpha_3$	$\delta\alpha_4$	$\delta\alpha_5$	$\delta\alpha_6$
1	1	1	1	-1					
2	3	2	3	-4	1				
3	11	6	11	-18	9	-2			
4	25	12	25	-48	36	-16	3		
5	137	60	137	-300	300	-200	75	-12	
6	147	60	147	-360	450	-400	225	-72	10

Table 5.5.1: Coefficients of the backward difference method (5.5.5c). for orders one through six [1]. Their local error coefficients are $1/(k+1)$.

5.6 Convergence, Accuracy, and Stability

Having several methods at our disposal, let us now study questions of consistency, convergence, and stability of linear multistep methods (LMMs) of the form (5.1.2) applied to the IVP (5.1.1). We begin by applying the usual definitions in our present situation.

Definition 5.6.1. *Let $z(t) \in C^1[0,T]$, $t_n - ih \in [0,T]$, $i = 0, 1, \ldots, k$, and*

$$\mathcal{L}_h(z(t)) = \frac{\sum_{i=0}^{k} \alpha_i z(t_n - ih)}{h} - \sum_{i=0}^{k} \beta_i z'(t_n - ih). \tag{5.6.1a}$$

The local discretization error of the LMM (5.1.2) is

$$\tau_n = \mathcal{L}_h(y(t_n)). \tag{5.6.1b}$$

Definition 5.6.2. *The local error is*

$$d_n = y(t_n) - y_n \tag{5.6.2}$$

assuming that no errors have occurred prior to t_n, i.e., $y_i = y(t_i)$, $i < n$.

Substituting exact solution values into (5.1.2) for $i = 0, 1, \ldots, n-1$, using (5.6.2), and using the normalization $\alpha_0 = 1$ (cf. (5.1.2c)), we find

$$y(t_n) - y_n = y(t_n) + \sum_{i=1}^{k} \alpha_i y(t_{n-i}) - h\beta_0 f(t_n, y_n) - h\sum_{i=1}^{k} \beta_i f(t_{n-i}, y(t_{n-i})).$$

Using (5.6.1)

$$y(t_n) - y_n = h\beta_0[f(t_n, y(t_n)) - f(t_n, y_n)] + h\tau_n.$$

Assuming that f is continuously differentiable and using the mean value theorem

$$f(t_n, y(t_n)) - f(t_n, y_n) = f_y(t_n, \xi_n)[y(t_n) - y_n]$$

where ξ_n is between y_n and $y(t_n)$. Using (5.6.2), we have

$$d_n = \frac{h\tau_n}{1 - h\beta_0 f_y(t_n, \xi_n)}.$$

Thus, the local error is proportional to the local discretization error for all LMMs and the local error is the product of the step size and the local discretization errors for all explicit LMMs ($\beta_0 = 0$).

Definition 5.6.3. *A LMM is of order p if p is the largest integer for which $\mathcal{L}_h(y(t_n)) = O(h^p)$.*

Definition 5.6.4. *A LMM is consistent if it is at least of order one.*

From Definition 5.6.4, consistency implies that the LMM must be exact when $y(t)$ is a linear polynomial (Problem 5.1.1). Using (5.1.2) with $y(t) = 1$ and $y(t) = t$ yields the explicit consistency conditions

$$\sum_{i=0}^{k} \alpha_i = 0, \qquad \sum_{i=1}^{k} i\alpha_i + \sum_{i=0}^{k} \beta_i = 0. \tag{5.6.3}$$

Definition 5.6.5. *Consider solving (5.1.1) using (5.1.2) on $0 < t \leq T$ with a sequence of meshes $\{0 = t_0 < t_1 < \ldots < t_N = T\}$ such that $N \to \infty$ and $h = \max_{1 \leq n \leq N}(t_n - t_{n-1}) \to 0$. A LMM is convergent if $y_n \to y(t_n)$, $n \in [0, N]$ uniformly as $N \to \infty$ ($h \to 0$).*

Stability and its relation to consistency and convergence is not as simple for LMMs as for one-step methods. While a convergent multistep method is necessarily consistent, consistency alone is not sufficient for convergence. Before tackling the general question of stability (absolute stability, *etc.*), let us consider a simple example.

Example 5.6.1. Consider the solution of the test problem

$$y' = \lambda y, \qquad y(0) = 1,$$

by the leap frog method

$$y_n = y_{n-2} + 2hf_{n-1} = y_{n-2} + 2h\lambda y_{n-1}, \qquad y_0 = 1.$$

A solution, calculated on $0 < t \leq 4.5$ with $\lambda = -2$ and $h = 1/32$, is shown in Figure 5.6.1. The solution appears to be correct initially, but oscillations of increasing amplitude develop and, at the very least, the solution is not absolutely stable. The local discretization error of the leap frog scheme is $O(h^2)$, so it is certainly consistent. Since the test equation is linear, it suffices to forgo perturbations and examine solution of

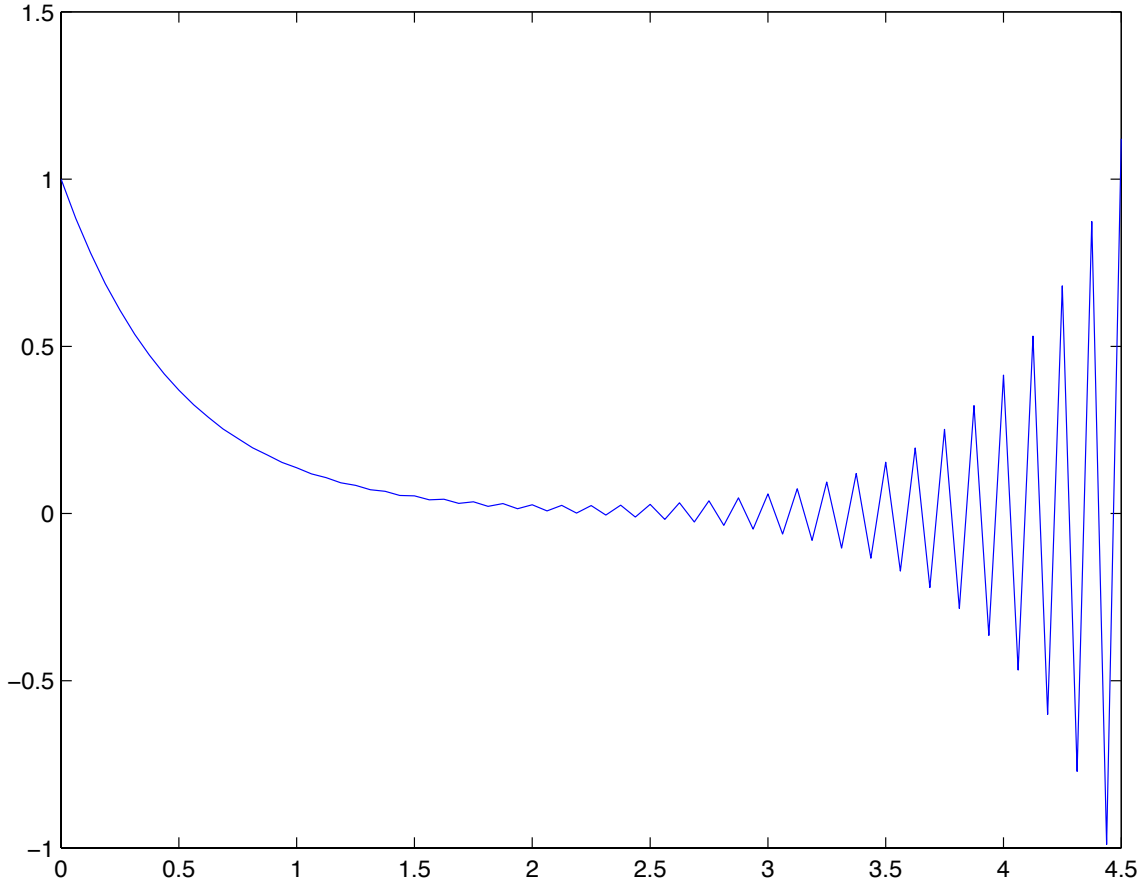

Figure 5.6.1: Solution of Example 5.6.1 with $\lambda = -2$ and $h = 1/32$.

$$y_n = y_{n-2} + 2h\lambda y_{n-1}, \qquad y_0 = \delta.$$

This is a second-order difference equation with constant (independent of n) coefficients. These difference equations have many similarities to second-order, constant-coefficient ODEs. In this case, the difference equation may be solved by assuming a solution of the form

$$y_n = c\xi^n.$$

Substituting this assumed solution into the leap frog difference equation yields

$$c\xi^n = c\xi^{n-2} + 2h\lambda c\xi^{n-1}.$$

Seeking non-trivial solutions, it is permissible to divide by $c\xi^{n-2}$ to obtain

$$\xi^2 - 2h\lambda\xi - 1 = 0.$$

The roots of this quadratic equation are

$$\xi_{1,2} = h\lambda \pm \sqrt{1 + (h\lambda)^2}.$$

The two roots satisfy the difference equation and the general solution is a linear combination of both of them; thus,

$$y_n = c_1\xi_1^n + c_2\xi_2^n.$$

The values of c_1 and c_2 are determined from the initial condition $y_0 = \delta$ and the starting value y_1; however, for our purposes, it won't be necessary to explicitly calculate c_1 and c_2. If, however, y_0 and y_1 are $O(\delta)$ then the c_1 and c_2 will also be $O(\delta)$.

Let's obtain more explicit information by assuming that $|h\lambda| \ll 1$. Then, expanding the square root,

$$\xi_1 = 1 + h\lambda + O((h\lambda)^2), \qquad \xi_2 = -[1 - h\lambda + O((h\lambda)^2)].$$

Thus,

$$y_n = c_1[1 + h\lambda + O((h\lambda)^2)]^n + (-1)^n c_2[1 - h\lambda + O((h\lambda)^2)]^n,$$

or

$$y_n = c_1 e^{\lambda hn} + (-1)^n c_2 e^{-\lambda hn} + O(n(h\lambda)^2).$$

The first term on the right approximates the exact ODE solution which, of course, is

$$y(t_n) = \delta e^{\lambda t_n} = \delta e^{\lambda hn}.$$

The second solution is called "parasitic" or "extraneous" and arises because the first-order ODE has been approximated by a second-order difference equation. If $\lambda > 0$, the parasitic solution

$$(-1)^n c_2 e^{-\lambda hn}$$

decays as n increases and causes little harm. However, if $\lambda < 0$, the parasitic solution grows with increasing n and eventually dominates the approximation of the exact solution. The parasitic solution oscillates from time step to time step as indicated by the presence of the $(-1)^n$ factor (Figure 5.6.1).

Let's continue our investigation of stability by re-stating the basic concept (Section 2.1) as it applies to a LMM.

Definition 5.6.6. *A LMM is stable if there exists an $\hat{h}$ for each $f(t, y)$ such that a change in the starting values by a fixed amount produces a bounded change in the numerical solution for all $h \in (0, \hat{h})$ and all $t \in [0, T]$.*

This definition is, once again, too general for practical verification. As usual, we rely on the test equation

$$y' = \lambda y, \qquad y(0) = \delta. \tag{5.6.4}$$

To conduct a stability analysis, we apply (5.6.4) to (5.1.2) to obtain

$$\sum_{i=0}^{k} (\alpha_i y_{n-i} - h\lambda\beta_i y_{n-i}) = 0. \tag{5.6.5}$$

This k *th* order constant-coefficient difference equation can be solved in the manner used for Example 5.6.1; thus, assume a solution of the form

$$y_n = c\xi^n \tag{5.6.6}$$

and substitute it into (5.6.5) to obtain

$$\sum_{i=0}^{k} (\alpha_i c\xi^{n-i} - h\lambda\beta_i c\xi^{n-i}) = 0.$$

Dividing by the common $c\xi^{n-k}$ factor, we find that ξ is a root of the polynomial

$$\rho(\xi) - h\lambda\sigma(\xi) = 0, \tag{5.6.7a}$$

where

$$\rho(\xi) = \sum_{i=0}^{k} \alpha_i \xi^{k-i}, \tag{5.6.7b}$$

and

$$\sigma(\xi) = \sum_{i=0}^{k} \beta_i \xi^{k-i}. \tag{5.6.7c}$$

Equations (5.6.7b) and (5.6.7c) are called the *first* and *second characteristic polynomials* of the LMM, respectively. Using (5.6.3), we see that the LMM is consistent if and only if

$$\rho(1) = 0, \qquad \rho'(1) - \sigma(1) = 0. \tag{5.6.7d}$$

Equation (5.6.7a) has k solutions, ξ_i, $i = 1, 2, \ldots, k$. One solution, say ξ_1, approximates the solution of the test problem (5.6.4). The remaining $k-1$ solutions are parasitic. If the roots of (5.6.7a) are distinct then the general solution of (5.6.5) is

$$y_n = \sum_{i=1}^{k} c_i \xi_i^n. \tag{5.6.8a}$$

The assumed solution (5.6.6) must be modified when some of the roots are equal. Suppose, for example, that ξ_i is a root of multiplicity μ, then the μ solutions corresponding to ξ_i are

$$(c_i + c_{i+1} n + c_{i+2} n^2 + \ldots + c_{i+\mu-1} n^{\mu-1}) \xi_i^n. \tag{5.6.8b}$$

Note the similarities between this analysis and that of constant-coefficient ODE [3].

Since the exact solution of the test equation (5.6.4) is

$$y(t_n) = \delta e^{h\lambda n} = \delta (e^{h\lambda})^n, \tag{5.6.9}$$

the principal root ξ_1 of (5.6.7) must be an approximation of $e^{h\lambda}$. If the remaining roots $|\xi_i| > 1$, $i = 2, 3, \ldots, k$, the corresponding parasitic solutions grow as n increases and the method, at the very least, cannot be absolutely stable. Let us emphasize that all roots ξ_i, $i = 1, 2, \ldots, k$, of (5.6.7a-c) depend continuously on the parameter $h\lambda$ by writing $\xi_i = \xi_i(h\lambda)$, $i = 1, 2, \ldots, k$. If $|\xi_i(0)| < 1$, $i = 1, 2, \ldots, k$, there is an $\hat{h}$ for any fixed λ such that $|\xi_i(h\lambda)| < 1$, $i = 1, 2, \ldots, k$, for $h < \hat{h}$. These arguments suggest that we can study the stability characteristics of the LMM by examining the roots of

$$\rho(\xi) = \sum_{i=1}^{k} \alpha_i \xi^i = 0, \tag{5.6.10}$$

which is the solution of (5.6.7a) when $h\lambda = 0$.

If the LMM is consistent then (5.6.7d) implies that unity is a root of (5.6.10). From (5.6.9) we see that this corresponds to the exact solution of the test equation, i.e.,

$$\xi_1(0) = 1. \tag{5.6.11}$$

We have noted that roots $\xi_i(0)$, $i = 2, 3, \ldots, k$, that are less than unity will remain so for sufficiently small values of $h\lambda$. Thus, the limiting stability (as $h\lambda \to 0$) is determined by those roots that are greater than or equal to unity in modulus. Any root having larger than unit modulus will produce unbounded solutions and must by unstable according to Definition 5.6.6. It, therefore, remains to investigate the behavior of those roots that have unit modulus when $h = 0$. Let ξ_i be such a root and let us initially assume that it is simple. Supposing that $|h\lambda| \ll 1$, let us expand $\xi_i(h\lambda)$ as the series in powers of $h\lambda$

$$\xi_i(h\lambda) = \xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2). \tag{5.6.12a}$$

Substituting (5.6.12a) into (5.6.7a)

$$\rho(\xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2)) - h\lambda\sigma(\xi_i(0) + \gamma_i h\lambda + O((h\lambda)^2)) = 0.$$

Expanding ρ and σ

$$\rho(\xi_i(0)) + h\lambda[\gamma_i\rho'(\xi_i(0)) - \sigma(\xi_i(0))] + O((h\lambda)^2) = 0.$$

Now, $\rho(\xi_i(0)) = 0$ according to (5.6.10). Thus, to satisfy (5.6.7a) for sufficiently small $h\lambda$, the coefficient of $h\lambda$ in the above expression must vanish. This yields

$$\gamma_i = \frac{\sigma(\xi_i(0))}{\rho'(\xi_i(0))}. \tag{5.6.12b}$$

Remark 1. The principal root $\xi_1(0) = 1$. This, with the consistency conditions (5.6.7d) further implies

$$\gamma_1 = \frac{\sigma(1)}{\rho'(1)} = 1. \tag{5.6.12c}$$

The series expansion suggests the existence of a constant c such that

$$|\xi_i(h\lambda)| \leq |\xi_i(0)|(1 + ch\lambda), \qquad h < \hat{h}.$$

Taking the nth power and recalling that $|\xi_i(0)| = 1$ yields

$$|\xi_i^n(h\lambda)| \leq |(1 + ch\lambda)^n| \leq |e^{ch\lambda n}| \leq |e^{c\lambda T}| \leq C, \qquad 0 \leq t \leq T, \qquad h < \hat{h}.$$

Therefore, the parasitic solutions only grow by a bounded amount for finite times when roots of unit magnitude are simple. Hence, the LMM is stable (but not absolutely stable) for the test equation in this case.

It remains to examine those roots of (5.6.10) that have unit magnitude but are not simple. Suppose that $\xi_i(0)$ is such a root with multiplicity μ, then, as indicated in (5.6.8b), the solution y_n would contain the terms $n\xi_i^n$, $n^2\xi_i^n$, $\ldots$, $n^{\mu-1}\xi_i^n$. With $|\xi_i(0)| = 1$, these terms are unbounded as $n \to \infty$. Hence, the LMM cannot be stable.

The preceding arguments suggest that the stability of a LMM is linked to the following condition.

Definition 5.6.7. *A LMM satisfies the root condition (or the condition of zero stability if the roots of $\rho(\xi) = 0$ are inside the unit circle or simple on the unit circle in the complex ξ-plane.*

Theorem 5.6.1. *Satisfaction of the root condition is necessary and sufficient for the stability of a LMM.*

Proof. The key aspect of the proof have been presented. A more thorough analysis is given in Gear [10], Section 10.1. □

Example 5.6.2. The leap frog method of Example 5.1.1 has $\alpha_0 = 1$ $\alpha_1 = 0$, and $\alpha_2 = -1$ with $k = 2$; thus, using (5.6.7b)

$$\rho(\xi) = \xi^2 - 1.$$

The roots of this equation are $\xi_1(0) = 1$ and $\xi_2(0) = -1$. Both roots have unit modulus and are simple on the unit circle in the complex plane; hence, the leap frog scheme satisfies the root condition and is stable.

While the leap frog scheme is stable, it is clear from Example 5.1.1 that solutions grow rapidly when they should be decaying. Clearly a finer distinction is needed and this prompts the concepts of strong and weak stability.

Definition 5.6.8. *A LMM is weakly stable if it is stable, but has more than one root of* $\rho(\xi) = 0$ *on the unit circle in the complex* ξ*-plane.*

Definition 5.6.9. *A LMM is strongly stable if all roots of* $\rho(\xi) = 0$ *are inside the unit circle in the* ξ*-plane except for the principal root* $\xi_1(0) = 1$*.*

Example 5.6.3. The results of Example 5.6.2 confirm that the leap frog scheme is weakly stable. Parasitic solutions of a weakly stable scheme may either grow or decay for nonzero values of $h\lambda$ as n increases, depending on whether they move outside or inside of the unit circle. In Example 5.6.1, we saw that the parasitic solution of the leap frog scheme grew when $h\lambda < 0$ and decayed when $h\lambda > 0$. All parasitic solutions of a strongly stable method decay for sufficiently small values of $h\lambda$ as n increases, since there modulus is less than unity when $h\lambda = 0$.

Example 5.6.4. Let us examine the stability of the Adams-Bashforth and Adams-Moulton methods as $h \to 0$. Recall (Sections 5.3 and 5.4) that the Adams' methods have the form (5.1.2) with $\alpha_0 = 1$, $\alpha_1 = -1$, and $\alpha_i = 0$, $i = 2, 3, \ldots, k$, i.e.,

$$y_n = y_{n-1} + h \sum_{i=0}^{k} \beta_i f_{n-i}.$$

The coefficient $\beta_0 = 0$ for the (explicit) Adams-Bashforth methods and is nonzero for the (implicit) Adams-Moulton methods.

Using the test equation (5.6.4)

$$y_n = y_{n-1} + h\lambda \sum_{i=0}^{k} \beta_i y_{n-i}.$$

Assuming that y_n is proportional to ξ^n leads to (5.6.7) which, in this case, is

$$\rho(\xi) - h\lambda\sigma(\xi) = 0,$$

$$\rho(\xi) = \xi^{k-1}(\xi - 1), \qquad \sigma(\xi) = \sum_{i=0}^{k} \beta_i \xi^{k-i}.$$

Thus, the roots of the first characteristic polynomial ($\rho(\xi) = 0$) are

$$\xi_1 = 1, \qquad \xi_2 = \xi_3 = \ldots = \xi_k = 0.$$

The parasitic roots $\xi_i(0)$, $i = 2, 3, \ldots, k$, are all zero for both the Adams-Bashforth and Adams-Moulton methods; thus, these methods are strongly stable.

Computation is performed at finite values of h. The root condition and the notions of strong and weak stability all involve reasoning in the limit of vanishingly small step sizes. This shortcoming encourages us to further extend concepts of stability.

Definition 5.6.10. *A LMM is absolutely stable for those values of $h\lambda$ where the roots of (5.6.7a) satisfy $|\xi(h\lambda)| \leq 1$ when applied to the test problem (5.6.4).*

Definition 5.6.11. *A LMM is relatively stable for those values of $h\lambda$ where the parasitic roots of (5.6.7) are less in magnitude than the principal root.*

The boundary of the region of absolute stability of a LMM may be calculated by the *boundary locus method* as described in Chapter 3 for one-step methods. Complex values of ξ having unit modulus are written in polar form

$$\xi = e^{i\theta}$$

and (5.6.7a) is imposed. The boundary of the region of absolute stability follows by calculating

$$h\lambda = \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})}, \qquad \theta \in [0, \pi].$$

The regions of absolute stability of the Adams-Bashforth, Adams-Moulton, and backward difference methods are illustrated in Figures 2, 3, and 4, respectively [1]. Some observations and conclusions follow.

1. The Adams-Moulton methods of orders 1 and 2 are the backward Euler method and the trapezoidal rule. The absolute stability regions of these methods are the exterior of a unit circle centered at $(1, 0)$ and the entire left-half of the $h\lambda$-plane, respectively. Neither region is shown in Figure 5.6.2. Ascher and Petzold's [1] definition of k differs from ours for Adams-Moulton methods. The regions labeled $k = 2$ to 4 in Figure 5.6.3 correspond to our $k = 3$ to 5.

2. Explicit methods always have a finite region of absolute stability.

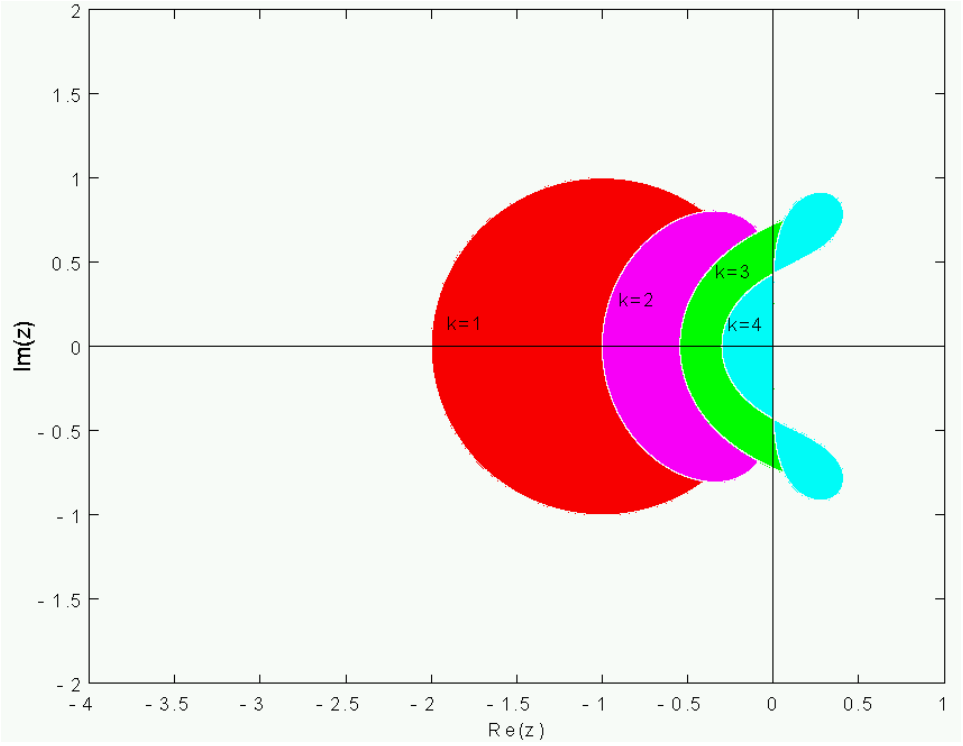

Figure 5.6.2: Absolute stability regions for Adams-Bashforth methods of orders 1 to 4. A method is stable inside of its shaded region [1].

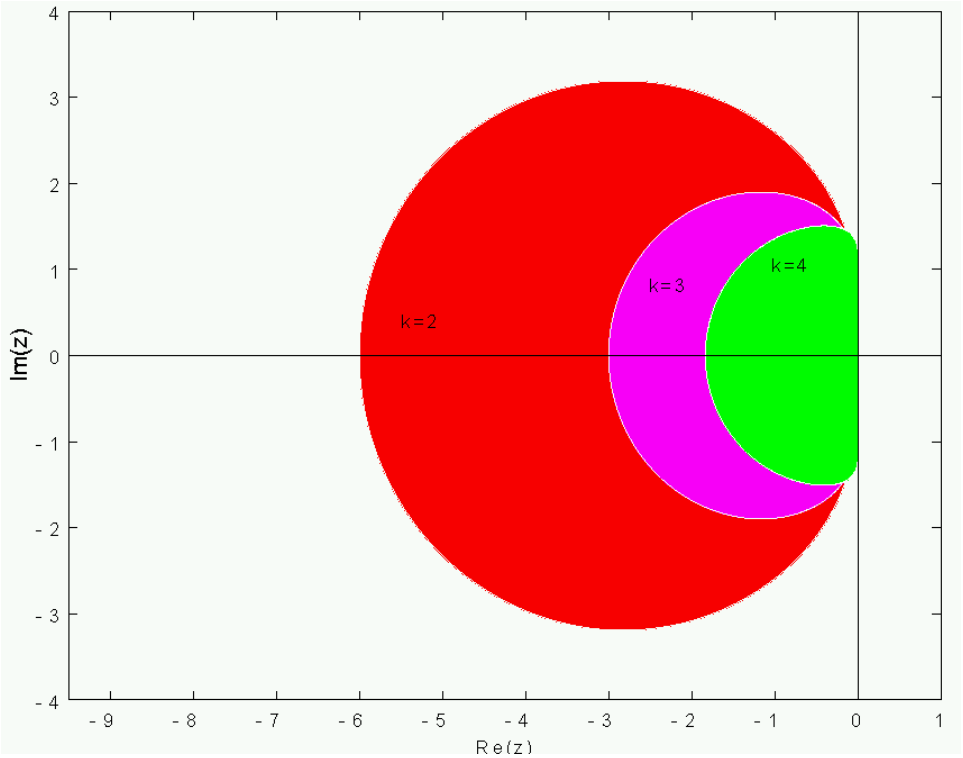

Figure 5.6.3: Absolute stability regions for Adams-Moulton methods of orders 3 to 5. A method is stable inside of its shaded region [1].

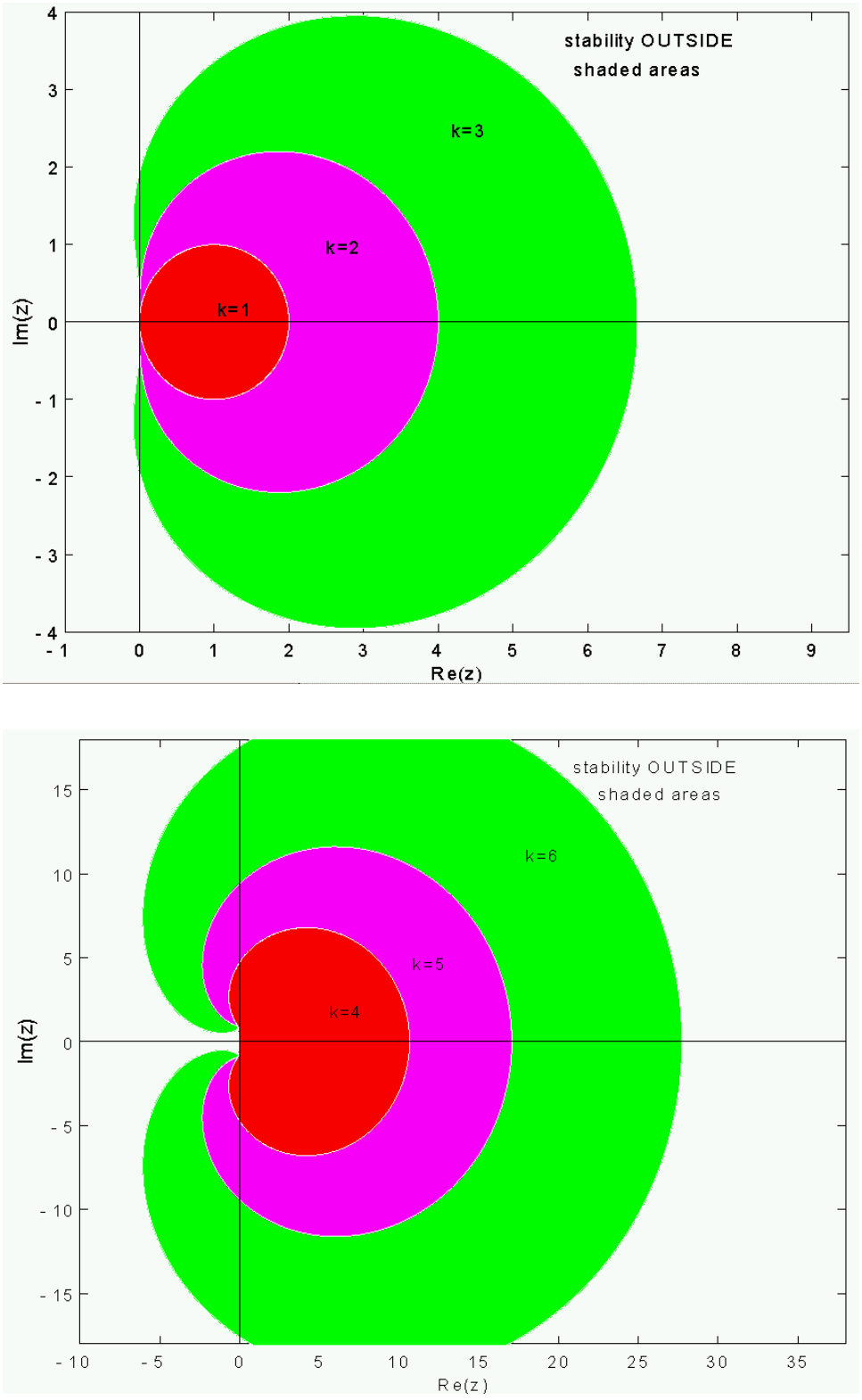

Figure 5.6.4: Absolute stability regions for backward difference formulas of orders 1 to 3 (top) and 4 to 6 (bottom). Methods are stable outside of the shaded regions [1].

3. Implicit methods have a larger region of absolute stability than a corresponding explicit method of the same order.

4. The absolute stability region typically becomes smaller as order increases.

5. The backward difference formulas shown in Figure 5.6.4 have unbounded absolute stability regions as $Re(h\lambda) \to -\infty$.

6. We have noted that the error coefficients of Adams-Moulton methods (5.4.5 - 5.4.8) are smaller than those of Adams-Bashforth methods (5.3.10 - 5.3.13) having the same order. Figures 5.6.2 and 5.6.3 indicate that the absolute stability regions of the Adams-Moulton methods are larger than those of the Adams-Bashforth methods of the same order. Therefore, the Adams-Moulton methods can be used with much larger step sizes than required for the Adams-Bashforth methods having the same accuracy. Typically, this increase in step size offsets the cost of solving an implicit difference equation.

The maximal order of a k-step LMM is $2k$ since there are $2k+1$ free parameters in the general formula (5.1.2); however, these maximal-order methods are typically unstable.

Theorem 5.6.2. *No k-step LMM satisfying the root condition can have order exceeding $k+1$ when k is odd or order $k+2$ when k is even.*

Proof. cf. Hairer *et al.* [12]. □

The k-step methods of order $k+2$ are called optimal-order methods. All roots of $\rho(\xi)$ are on the unit circle, so they are only weakly stable.

Example 5.6.5. The highest order two-step LMM

$$y_n = -4y_{n-1} + 5y_{n-2} + 2h(2f_{n-1} + f_{n-2})$$

has order four. Using (5.6.7b), the first characteristic polynomial for this method is

$$\rho(\xi) = \xi^2 + 4\xi - 5.$$

The roots of this polynomial, $\xi_1 = 1$ and $\xi_2 = -5$, do not satisfy the root condition; hence, the method is unstable. Let us illustrate this by applying the method to the very simple equation

$$y' = 0, \qquad y(0) = 0.$$

The exact solution of this IVP is, of course, the trivial solution $y(t) = 0$. For starting values, let us use

$$y_0 = 0, \qquad y_1 = \epsilon,$$

where ϵ represents a small rounding error.

Successive solutions satisfy

$$y_2 = -4y_1 + 5y_0 = -4\epsilon,$$

$$y_3 = -4y_2 + 5y_1 = 21\epsilon,$$

$$y_4 = -4y_3 + 5y_2 = -104\epsilon,$$

$$\ldots .$$

The step size h does not explicitly appear in the method for this IVP; hence, the solution cannot be bounded as $h \to 0$ and, therefore, the method is unstable. The method also does not converge as $h \to 0$; thus, consistency alone is not sufficient for convergence. Necessary and sufficient conditions for convergence are given in the following theorem which is due to G. Dahlquist.

Theorem 5.6.3. *The necessary and sufficient conditions for a LMM to be convergent are that it be consistent and satisfy the root condition.*

Proof. Complete proofs of this important theorem are given in Gear [10], Chapter 10, and Henrici [14], Chapter 5. We'll prove that convergence implies stability and consistency.

Applying the LMM (5.1.2) to the IVP $y' = 0$, $y(0) = 0$ yields

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = 0$$

Seeking to use a contradiction argument, suppose that $\rho(\xi)$ has a root ξ_2 with $|\xi_2| > 1$ and/or a root ξ_3 on the unit circle whose multiplicity exceeds unity. Further suppose

that the starting values are $O(h)$ (so that the starting values converge to $y_0 = 0$). The two solutions of the difference equation corresponding to ξ_2 and ξ_3 are

$$c_2 h \xi_2^n, \qquad c_3 hn \xi_3^n.$$

(The coefficients c_2 and c_3 have been scaled to reflect the $O(h)$ size of the solution.) The analysis is simplified by letting $Y(t)$ be a continuous function that interpolates to y_j, $j = 0, 1, \ldots$. Thus,

$$Y_2(t) = c_2 h \xi_2^{t/h}, \qquad Y_3(t) = c_3 t \xi_3^{t/h}, \qquad t = nh.$$

For fixed t, the solutions $Y_2(t)$ and $Y_3(t)$ are unbounded as $h \to 0$; thus, the solution cannot be convergent and we have a contradiction. Therefore, the LMM must be stable.

Next consider the IVP $y' = 0$, $y(0) = 1$, which has the solution $y(t) = 1$. With the present notation, the difference equation (5.1.2) becomes

$$\sum_{i=0}^{k} \alpha_i Y(t - ih) = 0.$$

Convergence implies that

$$\lim_{h \to 0} Y(t - ih) = 1$$

or, using (5.6.7b), that $\rho(1) = 0$.

Finally, consider the IVP $y' = 1$, $y(0) = 0$, which has the solution $y(t) = t$. The difference equation (5.1.2) for this problem is

$$\sum_{i=0}^{k} \alpha_i Y(t - ih) = h \sum_{i=0}^{k} \beta_i.$$

Convergence would imply that

$$Y(t - ih) = t - ih.$$

Substituting this function into the difference equation and using (5.1.2b) yields

$$\rho'(1) - \sigma(1) = 0,$$

which is the consistency condition (5.6.7d). Thus, both consistency conditions within (5.6.7d) are satisfied. □

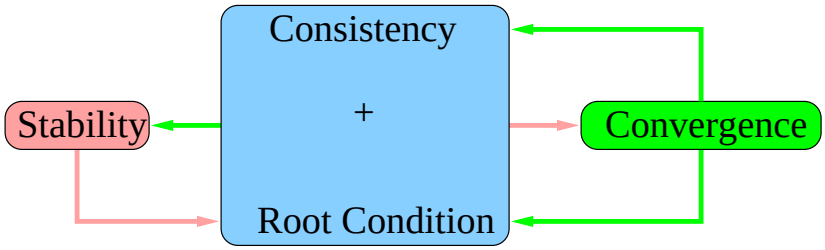

Figure 5.6.5: Relationship between consistency, convergence, the root condition, and stability.

A summary of the relationships between the basic concepts of consistency, convergence, and stability is given in Figure 5.6.5. Thus, from left to right, stability implies that the root condition is satisfied, which together with consistency implies convergence. Reading from right to left, convergence implies that the method is consistent and satisfies the root condition, which implies stability.

Let us recall the definition of A-stability introduced in Chapter 3.

Definition 5.6.12. *A LMM is A-stable if its region of absolute stability contains the left-half plane* $Re(h\lambda) \leq 0$.

Unfortunately, A-stability is very difficult to obtain for LMMs.

Theorem 5.6.4. *(i) No explicit LMM is A-stable, (ii) the order of an A-stable LMM cannot exceed two, and (iii) the second-order A-stable LMM with the smallest error coefficient is the trapezoidal rule.*

Proof. cf. Dahlquist [8]. A different proof of (ii) and (iii) appears in Wanner *et al.* [17]. □

This a major negative result of numerical ODEs. No such restriction exists for implicit Runge-Kutta methods and arbitrarily high-order A-stable methods may be constructed.

In order to obtain high orders of accuracy with LMMs, we will have to relax our stability demands and give up a portion of the left-half of the $h\lambda$-plane.

Definition 5.6.13. *A numerical method is* $A(\alpha)$*-stable,* $\alpha \in [0, \pi/2)$ *if its region of absolute stability contains the infinite wedge*

$$W_\alpha = \{h\lambda \mid \ -\alpha < \pi - arg(h\lambda) < \alpha\}.$$

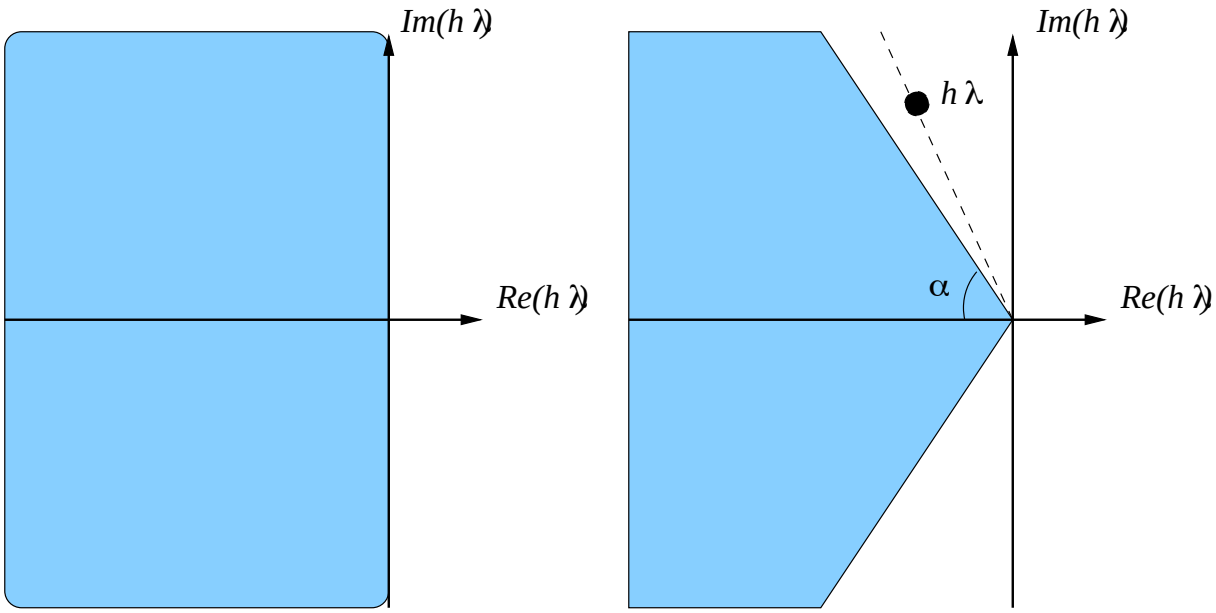

Figure 5.6.6: Regions of A-stability (left) and A(α)-stability (right).

The notion of A(α)-stability was introduced by Widlund [18] and is discussed in Gear [10], Chapter 11 and Hairer and Wanner [13], Section V.2. An example of the wedge W_α is shown in Figure 5.6.6. A(0^+)-stable methods have regions of absolute stability that contain the negative real axis. A($\pi/2$)-stable methods are A-stable. For a given λ with $Re(\lambda) < 0$, the point $h\lambda$ either lies inside W_α for all positive h or outside it for all positive h. If we know that all of the eigenvalues of a stiff system lie inside of a certain wedge W_{α^*}, then an A(α^*)-stable method can be used without any stability restriction on h.

Widlund [18] showed that A(α)-stable LMMs of orders one through four exist with α arbitrarily close to $\pi/2$. Grigorieff and Schroll [11] showed that k-step LMMs of order k also exist for all k (cf. [13], Section V.2).

Gear [10], Chapter 11, describes another means of relaxing A-stability by introducing the notion of stiff stability, which combines both stability and accuracy.

Definition 5.6.14. *A numerical method is stiffly stable if it is absolutely stable in the region* $R_1 = \{h\lambda \mid Re(h\lambda) \leq D_L\}$ *and accurate in the region* $R_2 = \{h\lambda \mid D_L < Re(h\lambda) < D_R,\ Im(h\lambda)| < D_I\}$.

The regions R_1 and R_2 are shown in Figure 5.6.7. Those eigenvalues λ that represent rapidly decaying terms in the transient solution of the ODE may be solved with step

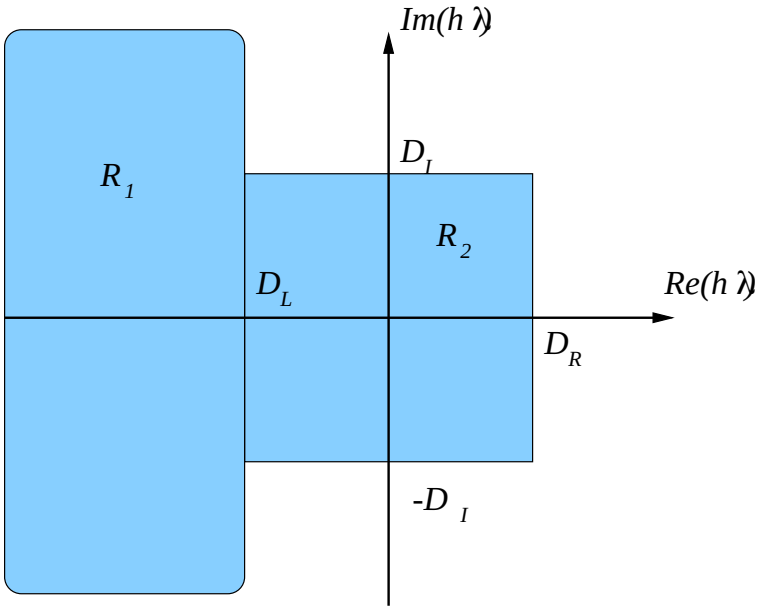

Figure 5.6.7: Schematic of a region of stiff stability.

sizes h such that $h\lambda \in R_1$. These components would be approximated stably but not accurately. Eigenvalues of the system representing slowly varying parts of the solution are smaller and their product with h would place them in R_2. Values of $h\lambda$ that are outside of $R_1 \cup R_2$ should not be used since either the growth of the solution is too fast (when $Re(h\lambda) > 0$) to be approximated accurately or the solution is too oscillatory (when $|Im(h\lambda)| > D_I$) to follow accurately. In both cases, h should be reduced so that $h\lambda \in R_2$.

Example 5.6.7. From Figure 5.6.4, we see that the backward difference formulas of orders one through six are stiffly stable. They are also A(α)-stable, but α decreases with increasing order.

Example 5.6.8. From (5.5.5), we recall that the backward difference formulas have the form

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = \beta_0 f_n.$$

Applying this method to the test equation (5.6.4) yields

$$\sum_{i=0}^{k} \alpha_i y_{n-i} - h\lambda\beta_0 y_n = 0.$$

Assuming a solution of the form ξ^n gives

$$\rho(\xi) - h\lambda\sigma(\xi) = 0,$$

with

$$\rho(\xi) = \sum_{i=0}^{k} \alpha_i \xi^{k-i}, \qquad \sigma(\xi) = \beta_0 \xi^k.$$

As $Re(h\lambda) \to -\infty$, the roots ξ satisfy

$$\sigma(\xi) = 0.$$

The roots of $\sigma(\xi) = 0$ are $\xi_i = 0$, $i = 1, 2, \ldots, k$. So all solutions of the backward difference formulas decay as $Re(h\lambda) \to -\infty$.

5.7 Implementation: Error and step size control

There are several implementation issues to discuss, including (i) starting values, (ii) iteration for implicit methods, (iii) error estimation, (iv) step-size variation, and (iv) method order variation. We'll begin with the solution of implicit difference equations, which are essential for stiff systems, but also preferred for non-stiff ODEs because of their smaller error coefficients and larger regions of absolute stability. Consider an implicit LMM of the form (5.1.2) and write it as

$$y_n + \sum_{i=1}^{k} \alpha_i y_{n-i} = h\beta_0 f(t_n, y_n) + h \sum_{i=1}^{k} \beta_i f_{n-i}, \tag{5.7.1}$$

where $\alpha_0 = 1$ (as a normalization), $\beta_0 \neq 0$, and y_{n-i} and f_{n-i}, $i = 1, 2, \ldots, k$, are assumed known. Equation (5.7.1) has a unique solution for h sufficiently small that may be computed by functional iteration

$$y_n^{(\nu)} = -\sum_{i=1}^{k} \alpha_i y_{n-i} + h\beta_0 f(t_n, y_n^{(\nu-1)}) + h \sum_{i=1}^{k} \beta_i f_{n-i}, \qquad \nu = 1, 2, \ldots, \tag{5.7.2a}$$

with $y_n^{(0)}$ being an initial guess for y_n. Convergence of the iteration occurs when h satisfies

$$hL|\beta_0| < 1 \tag{5.7.2b}$$

where L is a Lipschitz constant for $f(t,y)$. If $f \in C^1$ then we may take

$$L = \max_{y \in I} |f_y(t_n, y)|, \qquad I = [y_n - \delta, y_n + \delta]. \tag{5.7.2c}$$

For non-stiff problems, where L is small, the step size is usually determined by accuracy conditions rather than by (5.7.2b). Thus, functional iteration will give acceptable performance. However, for stiff problems, where L is large, the step size is severely restricted by (5.7.2b). Newtons iteration is typically provides better performance in these cases. With Newton's method, we find the roots of

$$F(y_n) = y_n + \sum_{i=1}^{k} \alpha_i y_{n-i} - h\beta_0 f(t_n, y_n) - h \sum_{i=1}^{k} \beta_i f_{n-i}. \tag{5.7.3a}$$

As noted in Section 2.2, Newton's method is obtained by linearizing (5.7.3a) about an iterate $y_n^{(\nu-1)}$; thus,

$$0 = F(y_n^{(\nu)}) = F(y_n^{(\nu-1)}) + F_y(y_n^{(\nu-1)})(y_n^{(\nu)} - y_n^{(\nu-1)}).$$

Using (5.7.3a) to calculate the derivative yields

$$[1 - h\beta_0 f_y(t_n, y_n^{(\nu-1)})](y_n^{(\nu)} - y_n^{(\nu-1)}) = -F(y_n^{(\nu-1)}), \qquad \nu = 1, 2, \ldots, \tag{5.7.3b}$$

Newton iteration converges when the initial guess $y_n^{(0)}$ is sufficiently close to y_n and/or h is sufficiently small. Unfortunately, it is difficult to get bounds like (5.7.2b,c) for Newton's method; however, typically h does not have to be as small as required for convergence with functional iteration (5.7.2). (Problem 1 contains a simple example.)

Newton's iteration usually converges when the Jacobian f_y is not reevaluated after each iteration. Thus, for example, we may use the same Jacobian for the entire iteration or even for iterations at several times.

With either functional or Newton iteration, a function evaluation must be made at each iteration. For vector systems, it is important to minimize the number of function evaluations and, thus, important to select an accurate initial guess $y_n^{(0)}$. Initial guess are usually chosen by a separate explicit method called a *predictor*. The implicit method is then called a *corrector*. For example, the Adams-Bashforth methods furnish predictors for use with Adams-Moulton correctors.

Two possible termination criteria for the corrector iteration are:

1. Iterate the corrector to convergence, e.g., terminate when

$$|y_n^{(\nu^*)} - y_n^{(\nu^*-1)}| \leq \epsilon,$$

where ϵ is on the order of the unit round off error of the computer. The local discretization and stability of this predictor-corrector combination are determined by the properties of the corrector alone; however, more function evaluations than necessary may be required.

2. Iterate the corrector a fixed number of times. This procedure reduces the number of function evaluations, but now the discretization error and stability characteristics of the result contain a mixture of the properties of both the predictor and corrector.

A compromise between these two strategies is to iterate the corrector a fixed number of times and to repeat the step with a smaller step size if adequate convergence was not obtained.

In describing predictor-corrector methods, we will use the notation

- P to denote an application of the predictor step,
- C to denote an application of the corrector step, and
- E to denote a function evaluation, i.e., an evaluation of f.

Example 5.7.1. A method that uses a predictor to evaluate $y_n^{(0)}$, does a function evaluation $f_n^{(0)} = f(t_n, y_n^{(0)})$, and one corrector iteration to obtain $y_n = y_n^{(1)}$ is a PEC method. If ν corrector iterations are performed and each iteration requires one evaluation of f, then the method is a $P(EC)^\nu$ method. In this case, the last function evaluation $f(t_n, y_n^{(\nu-1)})$ is saved as f_n for the next time step. Since we save $y_n = y_n^{(\nu)}$, it may be preferable to make an additional function evaluation and save $f_n = f(t_n, y_n^{(\nu)})$. This method then becomes a $P(EC)^\nu E$ method.

The orders of the predictor and corrector formulas need not be the same. Assume that the predictor has an order p and the corrector has an order q. Thus, the predictor satisfies

$$y_n^{(0)} = y(t_n) + O(h^{p+1})$$

and the corrector satisfies

$$y_n = y(t_n) + O(h^{q+1}).$$

The νth iteration of the corrector satisfies

$$y_n^{(\nu)} = y(t_n) + O(h^{p+1+\nu}) + O(h^{q+1}).$$

Thus, each corrector iteration decreases the effect of the predicted solution by one until the order of the corrector formula is reached. Since corrector iterations should be minimized, the order of the predictor is typically chosen as either q or $q-1$.

When the predictor and corrector formulas both have the same order q, the difference between them can be used to estimate the local error. Neglecting all errors prior to t_n and assuming the solution of the ODE to be sufficiently differentiable, the local error of the predicted and corrected solutions satisfy

$$y(t_n) - y_n^{(0)} = C_q^* h^{q+1} y^{(q+1)}(t_n) + O(h^{q+2})$$

and

$$y(t_n) - y_n^{(\nu)} = C_q h^{q+1} y^{(q+1)}(t_n) + O(h^{q+2}).$$

The constants C_q^* and C_q are known error coefficients of the predictor and corrector, respectively. Subtracting the above two equations to eliminate the exact solution $y(t_n)$ yields

$$(C_q^* - C_q) h^{q+1} y^{(q+1)}(t_n) = y_n^{(\nu)} - y_n^{(0)} + O(h^{q+2}).$$

Thus, for example, the local error of the corrected solution may be estimated as

$$C_q h^{q+1} y_n^{(q+1)} \approx \frac{C_q}{C_q^* - C_q}(y_n^{(\nu)} - y_n^{(0)}). \tag{5.7.4}$$

Example 5.7.2. Consider the use of the fourth-order Adams-Bashforth method (5.3.13),

$$y_n^{(0)} = y_{n-1} + \frac{h}{24}(55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}), \qquad d_n = \frac{251}{720}h^5 y^v(t_n) + O(h^6).$$

as a predictor and the fourth-order Adams-Moulton method (5.4.8),

$$y_n^{(\nu)} = y_{n-1} + \frac{h}{24}(9f_n^{(\nu-1)} + 19f_{n-1} - 5f_{n-2} + f_{n-3}), \qquad d_n = -\frac{19}{720}h^5 y^v(t_n) + O(h^6).$$

With $p = q = 4$, $C_4^* = 251/720$ and $C_4 = -19/720$. Using (5.7.4), the local error of the Adams-Moulton corrector may be estimated as

$$y(t_n) - y_n^{(\nu)} \approx -\frac{19}{270}(y_n^{(\nu)} - y_n^{(0)}).$$

Error estimates obtained by (5.7.4) are used to control step sizes so that specified local error tolerances are satisfied. Chosen step sizes must also satisfy appropriate (relative or absolute) stability conditions and ensure the convergence of the iterative scheme.

There is a second method of estimating the local errors of LMMs that does not rely on the predictor and corrector formulas having the same order. The development will also illustrate the appropriate information to be saved between time steps. We'll present the results in general but illustrate the approach when the third-order Adams-Bashforth method is used as a predictor and when the fourth-order Adams-Moulton method is as a corrector. Thus, from (5.3.12a) and (5.4.8a), we have

$$y_n^{(0)} = y_{n-1} + \frac{h}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3}) \tag{5.7.5a}$$

$$y_n^{(\nu)} = y_{n-1} + \frac{h}{24}(9f_n^{(\nu-1)} + 19f_{n-1} - 5f_{n-2} + f_{n-3}), \tag{5.7.5b}$$

Let us write these methods in a vector form by introducing

$$\mathbf{y}_{n-1} = \begin{bmatrix} y_{n-1} \\ hy'_{n-1} \\ hy'_{n-2} \\ hy'_{n-3} \end{bmatrix}, \qquad \mathbf{y}_n = \begin{bmatrix} y_n \\ hy'_n \\ hy'_{n-1} \\ hy'_{n-2} \end{bmatrix}, \tag{5.7.6a}$$

where $y'_n = f_n$, *etc.* The vector $\mathbf{y}_{n-1}$ contains the information needed to calculate y_n and $\mathbf{y}_n$ contains the information that is transferred to the next step. With this notation, the predicted solution (5.7.5a) is

$$\begin{bmatrix} y_n^{(0)} \\ hy_n^{'(0)} \\ hy'_{n-1} \\ hy'_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & \frac{23}{12} & -\frac{16}{12} & \frac{5}{12} \\ 0 & 3 & -3 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_{n-1} \\ hy'_{n-1} \\ hy'_{n-2} \\ hy'_{n-3} \end{bmatrix} \tag{5.7.6b}$$

or

$$\mathbf{y}_n^{(0)} = \mathbf{B}\mathbf{y}_{n-1}. \tag{5.7.6c}$$

The first row of (5.7.6b) is just the predictor formula (5.7.5a). The third and fourth rows are obvious identities. The second row can be developed from the Newton interpolation formula (5.3.5a), which we repeat here for convenience,

$$P_{k-1}(t) = \sum_{i=0}^{k-1} (-1)^i \begin{pmatrix} -\tau \\ i \end{pmatrix} \nabla^i y'_{n-1} \tag{5.7.7a}$$

where

$$\tau = \frac{t - t_{n-1}}{h} \tag{5.7.7b}$$

and $k = 3$ for the present example. Recall that $P_{k-1}(t)$ provides an approximation of $y'(t) = f(t, y(t))$ on the interval $[t_{n-k}, t_n]$; thus, setting $\tau = 1$ $(t = t_n)$ in (5.7.7a) yields

$$y_n^{'(0)} = P_{k-1}(t_n) = \sum_{i=0}^{k-1} (-1)^i \begin{pmatrix} -1 \\ i \end{pmatrix} \nabla^i y'_{n-1}.$$

But (cf. (5.3.4c,d))

$$(-1)^i \begin{pmatrix} -1 \\ i \end{pmatrix} = 1;$$

thus,

$$y_n^{'(0)} = \sum_{i=0}^{k-1} \nabla^i y'_{n-1}.$$

Setting $k = 3$ for the example at hand

$$y_n^{'(0)} = y'_{n-1} + (y'_{n-1} - y'_{n-2}) + (y'_{n-1} - 2y'_{n-2} + y'_{n-3})$$

or

$$y_n^{'(0)} = 3y'_{n-1} - 3y'_{n-2} + y'_{n-3}.$$

In a similar manner, we write the corrector as

$$\mathbf{y}_n^{(\nu)} = \mathbf{y}_n^{(\nu-1)} + \mathbf{c}\, g(\mathbf{y}_n^{(\nu-1)}), \qquad \nu = 1, 2, \ldots . \tag{5.7.8a}$$

The vector $\mathbf{c}$ and the scalar function $g(\mathbf{y}_n^{(\nu-1)})$ must be determined so that (5.7.8) agrees with the original corrector formula. We'll illustrate the procedure for the third-order corrector (5.7.5b). Setting $\nu = 1$ in (5.7.5b) and subtracting (5.7.5a) yields

$$y_n^{(1)} = y_n^{(0)} + \frac{3h}{8}[f(t_n, y_n^{(0)}) - 3y'_{n-1} + 3y'_{n-2} - y'_{n-3}].$$

From the above predicted interpolant, we have

$$y_n^{'(0)} - 3y'_{n-1} + 3y'_{n-2} - y'_{n-3} = 0.$$

Let us define $y_n^{'(1)} = f(t_n, y_n^{(0)})$ and add this to both sides of this relation to obtain

$$y_n^{'(1)} = y_n^{'(0)} + f(t_n, y_n^{(0)}) - 3y'_{n-1} + 3y'_{n-2} - y'_{n-3}.$$

The previous relations may be used for all iterates and not just for the first correction; thus, we obtain (5.7.8a) with

$$\mathbf{c} = \begin{bmatrix} 3/8 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \tag{5.7.8b}$$

$$g(\mathbf{y}_n^{(\nu-1)}) = h[f(t_n, y_n^{(\nu-1)}) - 3y'_{n-1} + 3y'_{n-2} - y'_{n-3}]. \tag{5.7.8c}$$

The form of the predictor-corrector pair given by (5.7.6c) and (5.7.8) avoids saving y_{n-1}, y_{n-2}, and y_{n-3}. Functional iteration is assumed. The formulas would have to be modified for Newton iteration. Our development does not reveal the generality of (5.7.6c) and (5.7.8). Gear [10], Chapters 7 and 9, provides additional details. The final converged iterates are passed to the next step; thus, if convergence occurs after ν corrector iterations, $\mathbf{y}_n = \mathbf{y}_n^{(\nu)}$.

The representation (5.7.6c) and (5.7.8) may not be best when automatic step and order changes are involved. Formulas involving backward differences may, for example, be best for these cases. Having the general representation (5.7.6c) and (5.7.8), we can switch to another one by a linear transformation. Thus, let

$$\mathbf{a}_n = \mathbf{T}\mathbf{y}_n \tag{5.7.9a}$$

and use (5.7.6b) and (5.7.8) to get a new predictor-corrector pair

$$\mathbf{a}_n^{(0)} = \mathbf{A}\mathbf{a}_{n-1}, \tag{5.7.9b}$$

$$\mathbf{a}_n^{(\nu)} = \mathbf{a}_n^{(\nu-1)} + \mathbf{l}P(\mathbf{a}_n^{(\nu)}) \tag{5.7.9c}$$

where

$$\mathbf{A} = \mathbf{TBT}^{-1}, \qquad \mathbf{l} = \mathbf{Tc}, \qquad P(\mathbf{a}_n) = g(\mathbf{T}^{-1}\mathbf{a}_n). \tag{5.7.9d}$$

Example 5.7.3. Let us develop the backward-difference form of the Adams predictor-corrector pair (5.7.5a,b). Let

$$\mathbf{a}_n = \begin{bmatrix} y_n \\ hy'_n \\ \nabla hy'_n \\ \nabla^2 hy'_n \end{bmatrix}.$$

Using the definition of the backward-difference operators (5.2.15)

$$\begin{bmatrix} y_n \\ hy'_n \\ \nabla hy'_n \\ \nabla^2 hy'_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} y_n \\ hy'_n \\ hy'_{n-1} \\ hy'_{n-2} \end{bmatrix} = \mathbf{T}\mathbf{y}_n.$$

Nordsieck [15] suggested representing the solution in terms of scaled derivatives at $t = t_n$, i.e., as

$$\mathbf{a}_n = [y_n, hy'_n, \frac{h^2}{2!}y''_n, \ldots, \frac{h^{k-1}}{(k-1)!}y_n^{(k-1)}]^T. \tag{5.7.10}$$

Step size variation is simplified with this representation. The transformation between representations may be obtained from the Newton backward-difference polynomial (5.3.3a)

$$P_{l-1}(t) = \sum_{i=0}^{l-1}(-1)^i \begin{pmatrix} -\tau+1 \\ i \end{pmatrix} \nabla^i y'_n. \tag{5.7.11}$$

(The relationship between the order k and the degree of the Newton polynomial $l-1$ will become clear shortly.) Since $P_{l-1}(t)$ is an approximation of $y'(t)$, differentiating (5.7.11) and setting $\tau = 1$ gives a relation between the Nordsieck and backward-difference representations. A similar linear relationship between the backward-difference and standard representation (5.7.6b) can be used to relate the Nordsieck and the standard variables.

Example 5.7.4. We'll construct a relationship between the Nordsieck and standard representations for the three-step (fourth-order) Adams method (5.7.5b). The Nordsieck vector for this method is

$$\mathbf{a}_n = [y_n,\ hy'_n,\ \frac{h^2}{2}y''_n,\ \frac{h^3}{6}y'''_n]^T.$$

A second-degree polynomial suffices to get the necessary derivatives; thus, setting $l = 3$ in (5.7.11), we have

$$P_2 = y'_n + (\tau - 1)\nabla y'_n + \frac{1}{2}(\tau - 1)\tau\nabla^2 y'_n.$$

Differentiating while using (5.7.7b)

$$h\frac{dP_2(t)}{dt} = \frac{dP_2(\tau)}{d\tau} = \nabla y'_n + \frac{1}{2}(2\tau - 1)\nabla^2 y'_n.$$

Setting $\tau = 1$ ($t = t_n$) gives an approximation of y''_n as

$$hy''_n = \nabla y'_n + \frac{1}{2}\nabla^2 y'_n.$$

Differentiating again

$$h^2\frac{d^2P_2(t)}{dt^2} = \frac{d^2P_2(\tau)}{d\tau^2} = \nabla^2 y'_n.$$

Setting $\tau = 1$

$$h^2 y'''_n = \nabla^2 y'_n.$$

Summarizing the results in matrix form

$$\begin{bmatrix} y_n \\ hy'_n \\ h^2y''_n/2 \\ h^3y'''_n/6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/2 & 1/4 \\ 0 & 0 & 0 & 1/6 \end{bmatrix} \begin{bmatrix} y_n \\ hy'_n \\ \nabla hy'_n \\ \nabla^2 hy'_n \end{bmatrix}.$$

Multiplying by the transformation of Example 5.7.3 gives the desired relationship between the Nordsieck and traditional forms as

$$\begin{bmatrix} y_n \\ hy'_n \\ h^2y''_n/2 \\ h^3y'''_n/6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3/4 & -1 & 1/4 \\ 0 & 1/6 & -1/3 & 1/6 \end{bmatrix} \begin{bmatrix} y_n \\ hy'_n \\ hy'_{n-1} \\ hy'_{n-2} \end{bmatrix}.$$

Example 5.7.5. Storage was at a premium when the above transformations were developed. Now, with a third-degree interpolating polynomial and less pressure on storage, we may store an extra derivative. Considering (5.7.5b) once again, let us approximate the solution derivative more accurately using a third-degree polynomial. Thus, using (5.7.11) with $l = k = 4$

$$P_3 = y'_n + (\tau - 1)\nabla y'_n + \frac{1}{2}(\tau - 1)\tau\nabla^2 y'_n + \frac{1}{6}(\tau - 1)\tau(\tau + 1)\nabla^3 y'_n.$$

Differentiating while using (5.7.7b)

$$h\frac{dP_3(t)}{dt} = \frac{dP_3(\tau)}{d\tau} = \nabla y'_n + \frac{1}{2}(2\tau - 1)\nabla^2 y'_n + \frac{1}{6}(3\tau^2 - 1)\nabla^3 y'_n.$$

Setting $\tau = 1$

$$hy''_n = \nabla y'_n + \frac{1}{2}\nabla^2 y'_n + \frac{1}{3}\nabla^3 y'_n.$$

Differentiating again

$$h^2\frac{d^2P_3(t)}{dt^2} = \frac{d^2P_3(\tau)}{d\tau^2} = \nabla^2 y'_n + \tau\nabla^3 y'_n.$$

Setting $\tau = 1$

$$h^2 y'''_n = \nabla^2 y'_n + \nabla^3 y'_n.$$

Differentiating once again

$$h^3 y^{iv}_n = \nabla^3 y'_n.$$

Thus, enlarging the Nordsieck vector by one, we have

$$\begin{bmatrix} y_n \\ hy'_n \\ h^2y''_n/2 \\ h^3y'''_n/3! \\ h^4y^{iv}_n/4! \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/4 & 1/6 \\ 0 & 0 & 0 & 1/6 & 1/6 \\ 0 & 0 & 0 & 0 & 1/24 \end{bmatrix} \begin{bmatrix} y_n \\ hy'_n \\ \nabla hy'_n \\ \nabla^2 hy'_n \\ \nabla^3 hy'_n \end{bmatrix}.$$

We've already seen that the local error of a k *th*-order multistep method has the form

$$d_n = C_k h^{k+1} y^{(k+1)}(t_n) + O(h^{k+2}). \tag{5.7.12a}$$

The Nordsieck vector (5.7.10) contains approximations of y_n and its first $k-1$ derivatives. However, by enlarging it to store the k *th* derivative (Example 5.7.5), we can construct an error estimate. Thus, let

$$\mathbf{a}_n = [y_n,\ hy'_n,\ \frac{h^2}{2!}y''_n,\ \ldots,\ \frac{h^k}{k!}y^{(k)}_n]^T \tag{5.7.12b}$$

and take a backward difference at two consecutive time steps to obtain

$$\nabla\mathbf{a}_n = \mathbf{a}_n - \mathbf{a}_{n-1}.$$

The last component of this vector $\nabla a_{n,k+1}$ is an approximation of $h^{k+1}y^{(k+1)}(t_n)/k!$. The local error can, thus, be approximated as

$$d_n \approx C_k k! \nabla a_{n,k+1}. \tag{5.7.12c}$$

Example 5.7.6. The local discretization error of the fourth-order Adams-Moulton method (5.7.5b) is given by (5.4.8b) as

$$d_n = -\frac{19}{720}h^5 y^v(t_n) + O(h^5).$$

Differencing the last component of the Nordsieck vector of Example 5.7.5 at t_n and t_{n-1} gives

$$\nabla a_{n,5} = \frac{h^4}{24}(y_n^{iv} - y_{n-1}^{iv}) = \frac{h^5}{24}\frac{y_n^{iv} - y_{n-1}^{iv}}{h} \approx \frac{h^5}{24}y^v(t_n).$$

Substituting into the discretization error formula

$$d_n \approx -\frac{19}{30}\nabla a_{n,5}.$$

Suppose we seek to keep $|d_n| \approx \epsilon$, then, either upon completion of a successful step or upon failure of a step, we can use (5.7.12a) to calculate a new step size αh such that

$$|C_k(\alpha h)^{k+1}y^{(k+1)}(t_n)| \approx \epsilon$$

or, using (5.7.12c),

$$|C_k k! \alpha^{k+1} \nabla a_{n,k+1}| \approx \epsilon.$$

Thus,

$$\alpha \approx \left(\frac{\epsilon}{|C_k k! \nabla a_{n,k+1}|}\right)^{1/(k+1)}. \tag{5.7.13a}$$

Knowing that the local error of the next (higher-order) method of a sequence of methods is

$$d_n = C_{k+1}h^{k+2}y^{(k+2)}(t_n) + O(h^{k+2}),$$

we can also calculate an approximation of $y^{(k+2)}(t_n)$ that can be used to change orders. Using second-order backward differences of the Nordsieck vector

$$\nabla^2 a_{n,k+1} \approx \frac{h^{k+2}y^{(k+2)}(t_n)}{k!}.$$

and

$$d_n \approx C_{k+1} k! \nabla^2 a_{n,k+1}. \tag{5.7.13b}$$

Estimates of the error of the next lower-order method follow directly from the next-to-last entry in the Nordsieck array. Thus, suppose that the error formula for the $k-1$-order method is

$$d_n = C_{k-1} h^k y^{(k)}(t_n) + O(h^{k+1}).$$

Then, (cf. Example 5.7.5 and (5.7.12b))

$$a_{n,k-1} = \frac{h^{k-1}}{(k-1)!} y_n^{(k-1)}$$

and

$$d_n \approx C_{k-1}(k-1)! h a_{n,k-1}. \tag{5.7.13c}$$

Formulas (5.7.13b,c) can be used to increase or decrease the method order by one. These order variations can be combined with step-size variations to produce a LMM code capable of both step and order adjustments. Most LMM codes do this according to the following guidelines.

1. Start the code with a first-order method. For the Adams methods, this would be the Euler-backward Euler pair. This avoids the need to incorporate separate one-step (Runge-Kutta) software in the code.

2. Change order before step size. Changing order is generally much more efficient than changing step size. Order increases and decreases in unit amounts can proceed by examining changes in the sequence of derivatives $y_n^{(k-1)}$, $y_n^{(k)}$, and $y_n^{(k+1)}$. These derivatives may be computed as described for (5.7.13). If the sequence of derivatives are decreasing in magnitude then computational reductions can be achieved by increasing the method order from k to $k+1$. On the contrary, if the sequence of derivatives are increasing, a reduction of order may be appropriate.

3. Include heuristics to avoid increasing the order too often. One possibility is to do at least k steps with a method of order k before considering an order change.

4. Starting values when changing step size or order may be computed by a Taylor's series when using the Nordsieck representation (5.7.10).

There are also variable step size LMMs ([12], Section III.5). The coefficients of these formulas are functions of h. The variable step formulas are generally more stable than the uniform step formulas but are less efficient.

Some available LMM codes are

1. *DEABM* is a modification of a code developed by Shampine and Gordon [16]. This code implements a variable step size divided difference representation of the Adams formulas. It uses a $PECE$ strategy and includes order variation. Let's go over the order variation scheme to illustrate the technique. After performing a step with an order k method, compute estimates $\mathbf{d}_n^{k-2}$, $\mathbf{d}_n^{k-1}$, and $\mathbf{d}_n^k$ of the local error of solutions with methods of order $k-2$, $k-1$, and k, respectively. Reduce the order to $k-1$ if

$$\max[\|\mathbf{d}_n^{k-2}\|, \|\mathbf{d}_n^{k-1}\|] \leq \|\mathbf{d}_n^k\|. \tag{5.7.14}$$

   Increase the order when a step is successful, (5.7.14) is violated, and a constant step size is used. (Remember, these are variable step-size methods.) Estimates of local discretization errors are obtained using approaches similar to (5.7.13). Norms are used for vector systems.

2. *EPISODE*, developed by Byrne and Hindmarsh [7], is a variable step, variable order implementation of the constant-step Adams and BDF methods using the Nordsieck representation. For nonstiff problems, functional iteration uses a $P(EC)^\nu$ strategy. Newton's method is used for stiff problems.

3. *LSODE* is another implementation of the constant-step Adams and BDF methods. It is similar to EPISODE.

4. *VODE* is a variable step, variable order code based on the variable-step Adams and BDF formulas. It was developed by Brown *et al.* [5] and is an extension of *EPISODE*.

Code	Symbol	Storage/eqn.
DEABM	- - - -	22
EPISODE	-..-..-	18
LSODE	-.-.-.-	17
D02CAF	... .	19
DOPRI8	– – – – –	9

Table 5.7.1: Legends for Figures 5.7.1 and 5.7.2 and code storage [12].

5. *DASSL*, developed by Petzold (cf. [4]) is a backward difference code for stiff differential and differential algebraic systems. DASSL addresses differential equations in the implicit form

$$\mathbf{f}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}.$$

It can also solve these systems when the Jacobian $\mathbf{f}_{\mathbf{y}'}$ is not invertible. In these cases, $\mathbf{y}'$ cannot be determined and the problem is a system of differential-algebraic equations. A simple two-dimensional example is

$$y_1' = a_{11}y_1 + a_{12}y_2 + b_1,$$

$$0 = a_{21}y_1 + a_{22}y_2 + b_2.$$

Many of these codes have been incorporated into scientific subroutine libraries. For example, the *IVPAG* procedure in the *IMSL* Library is a modification of Gear's [10] original Adam's and BDF procedure. The procedures *D02PCF* and *D02EJF*, respectively, are variable order, variable step Adams and BDF codes in the *NAG* Library.

Example 5.7.7. Hairer *et al.* [12], Section III.7, compare several codes on a suite of six non-stiff problems. We report their results in Figures 5.7.1 and 5.7.2. Codes that have not already been identified include *DOPRI8*, which is the eighth-order Dormand and Prince Runge-Kutta method [9], and *D02CAF*, which is similar to *LSODE* and *EPISODE*, and contained in the *NAG* library. The six problems used for testing are described in Hairer *et al.* [12], Section II.10. The legends and storage requirements for each code are given in Table 5.7.1.

The Runge-Kutta code DOPRI8 generally uses more function evaluations but less CPU time than the LMM codes. The test problems are fairly simple, but the performance

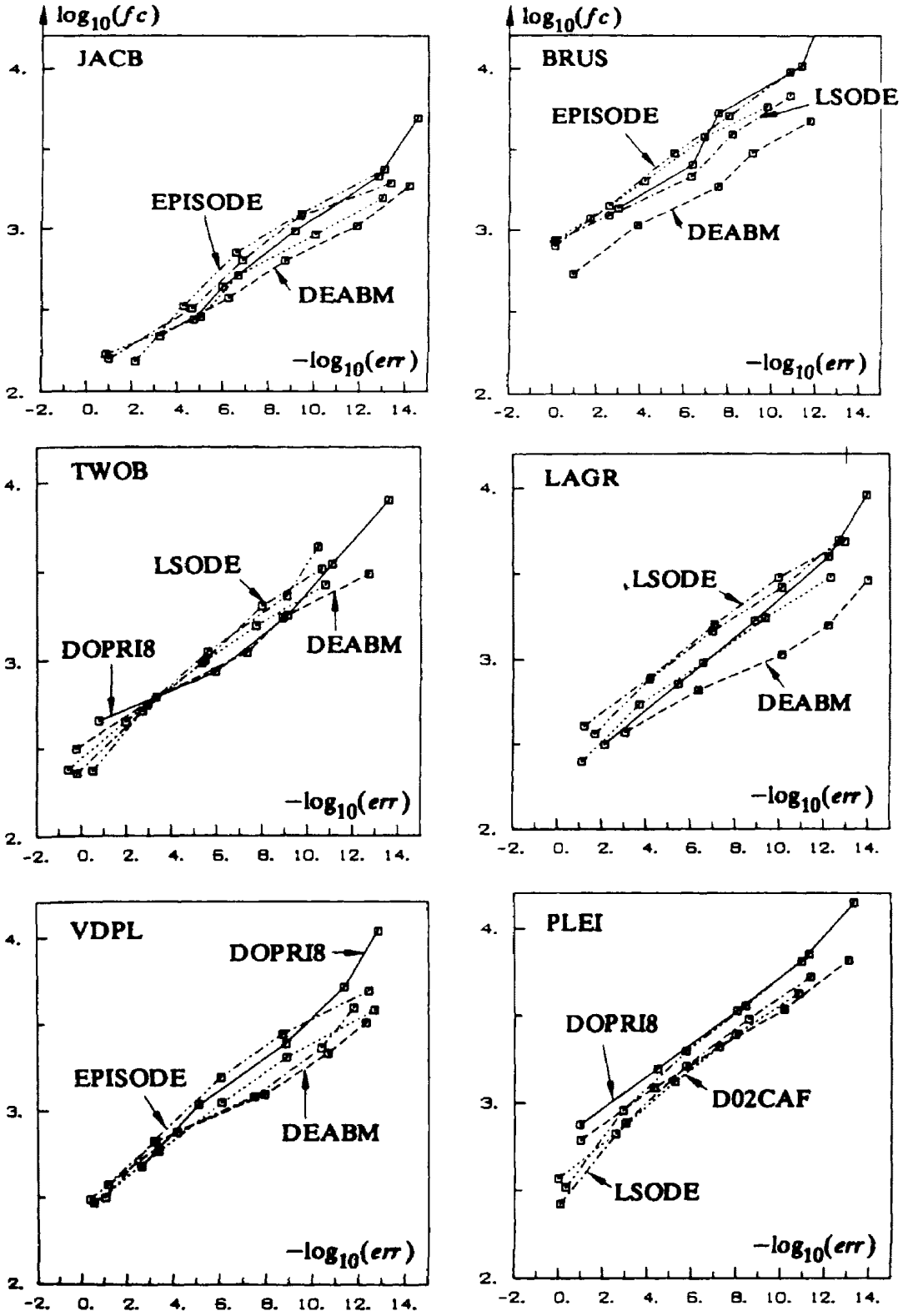

Figure 5.7.1: Number of function evaluations for a six-problem suite of non-stiff problems [12].

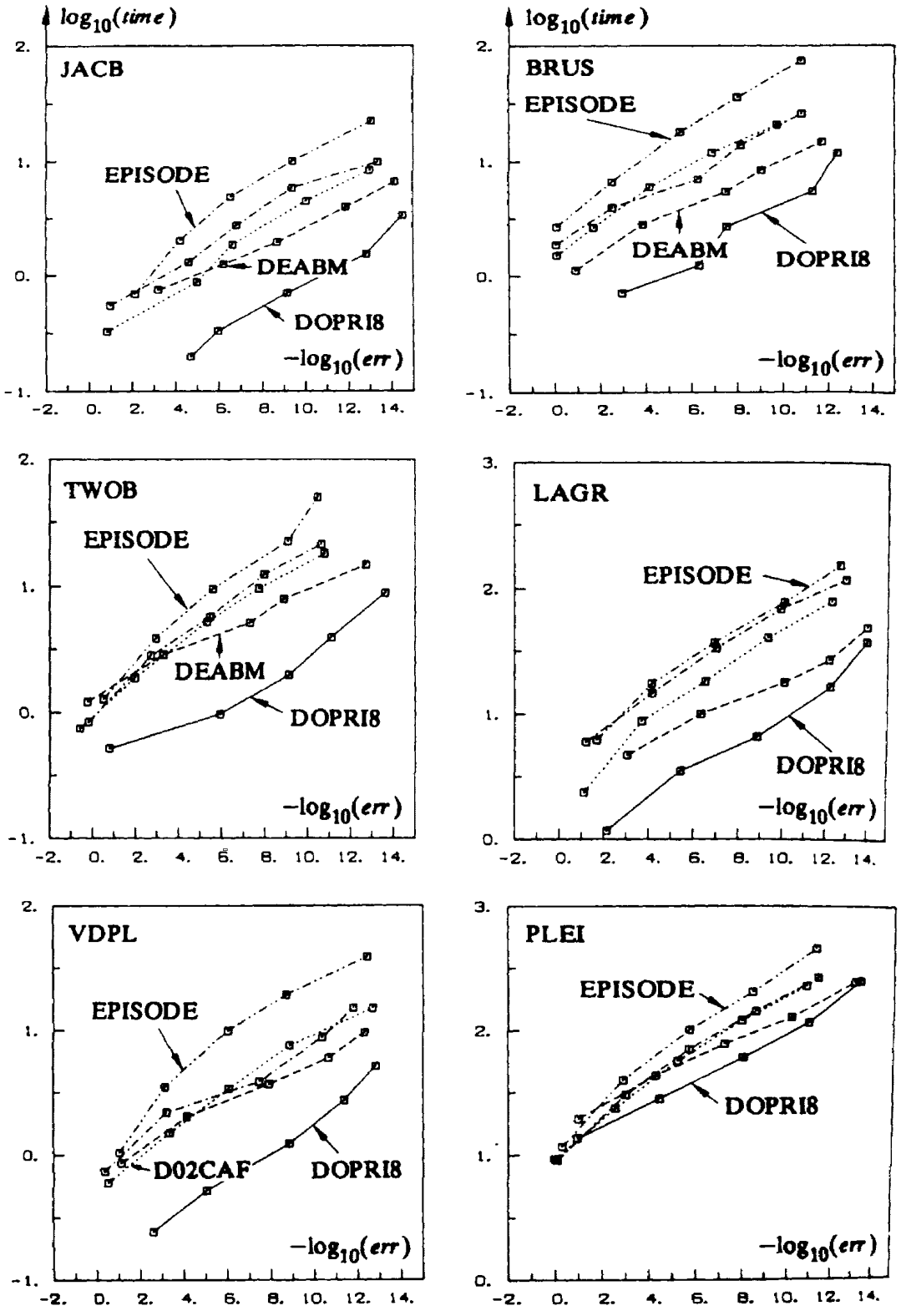

Figure 5.7.2: CPU times (seconds) for a six-problem suite of non-stiff problems [12].

of the high-order Runge-Kutta method should not be ignored. Conditions, however, could be reversed with more complicated functions. Of the LMM codes, *DEABM* seems to be the most efficient and *EPISODE* the least. From Table 5.7.1, we see that DOPPI8 requires about half of the storage per equation as do the LMMs. Of the LMM codes, DEBEAM has the greatest storage cost.

Example 5.7.8. Hairer and Wanner [13], Section V.5, compare several codes for a suite of stiff problems. We report their results in Figures 5.7.3 and 5.7.4. The codes used for this test follow.

1. *LSODE* and *VODE* are the codes described previously with the BDF option set.

2. *DEBDF* is a driver for *LSODE* for stiff systems.

3. *SPRINT*, developed by Berzins and Furzeland [2], contains several multistep methods and solution packages. The one used for the test is a blended multistep method that is A-stable to order four.

4. *SECDER* and *ROBER* are multistep codes that won't be described further.

5. *LADAMS* is *LSODE* with the Adams methods selected.

6. *RADAU5* is a fifth-order implicit Runge-Kutta method based on collocation at Radau points.

The results are scattered and difficult to interpret. The implicit Runge-Kutta method appears to do reasonably well on the small stiff problems of Figure 5.7.3 but less well on the larger problems of Figure 5.7.4. The Adams software did well on several of the smaller stiff problems but had difficulties with the larger problems. The opposite appears true for *VODE*. The SPRINT software also did well on several of the larger problems.

Some final notes of comparison between Runge-Kutta and LMMs follow:

1. Runge-Kutta methods are preferred to Adams methods when function evaluations are inexpensive.

2. In general, the stability of an Adams predictor-corrector pair is better than that of an explicit Runge-Kutta method of the same order.

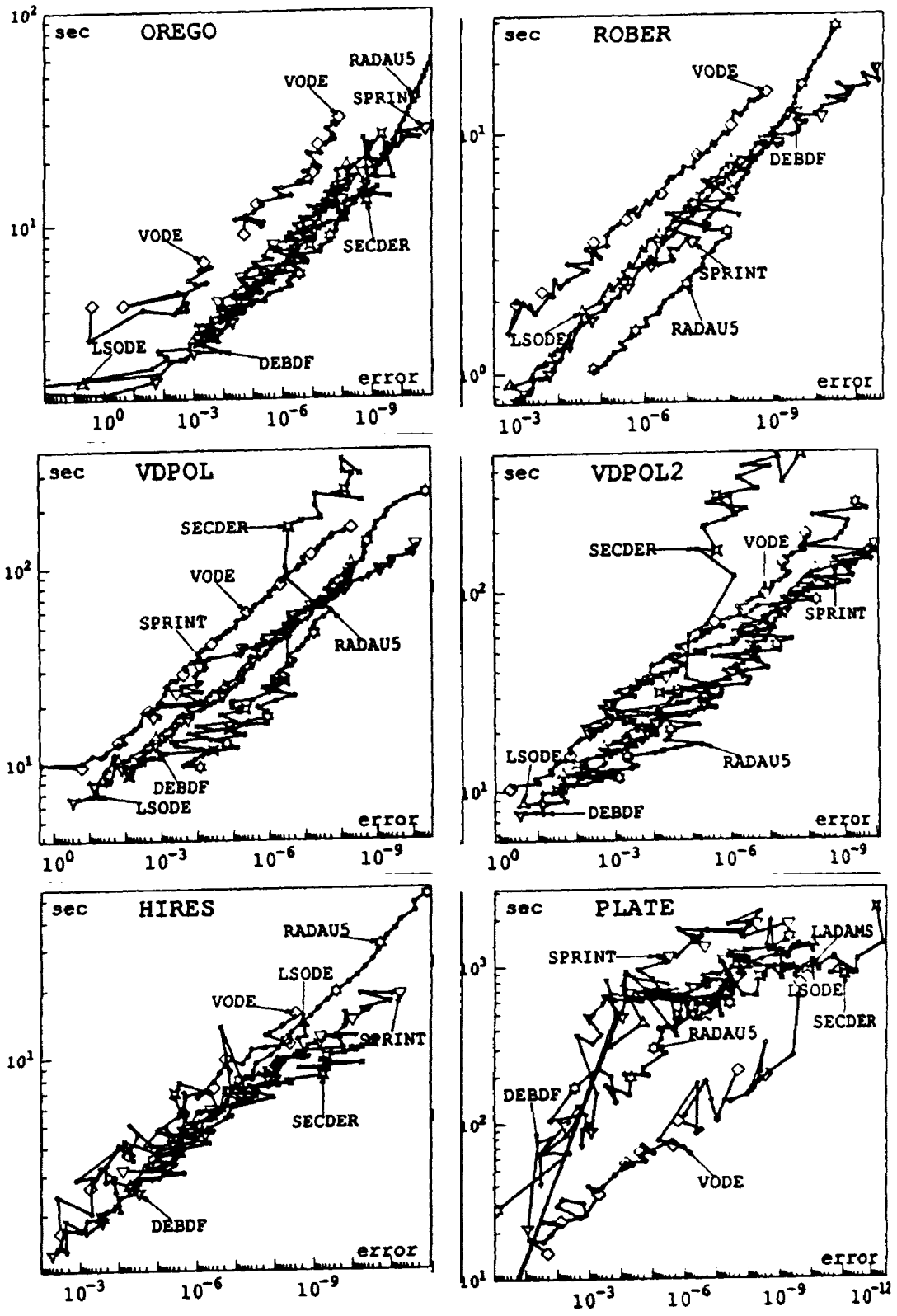

Figure 5.7.3: CPU time vs. error for a suite of small stiff problems [13].

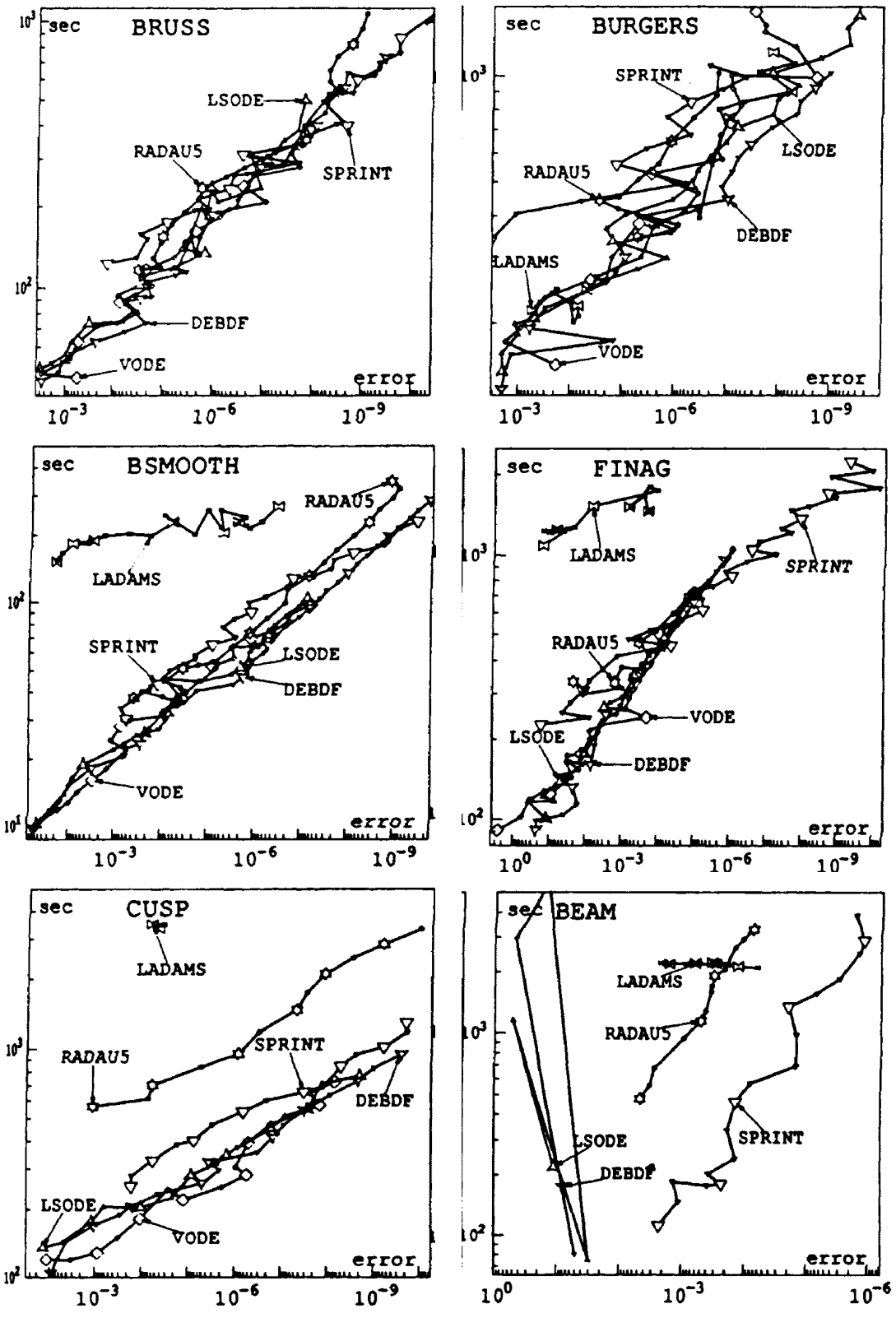

Figure 5.7.4: CPU times vs. error for a suite of large stiff problems [13].

3. Estimation of local errors is less expensive with a LMM than with a Runge-Kutta method. As seen in Examples 5.7.7 and 5.7, however, Runge-Kutta methods remain competitive with LMMs as long as the function evaluations aren't too expensive.

4. Fixed-order Runge-Kutta methods are easier to implement than LMMs. However, good software of both types exist.

5. For large stiff problems, BDFs are much more efficient than implicit Runge-Kutta methods unless the differential system has rapidly oscillating solutions. The solution of the BEAM problem of Example 5.7.4 is oscillatory and many BDF codes failed on it. The Runge-Kutta code *RADAU5* was the second-most efficient method on this problem. The A-stable method within the SPRINT package was, by far, the most efficient technique.

Problems

1. Consider the solution of

$$y' = \lambda y + y^2, \qquad t > 0, \qquad y(0) = 1,$$

which has the exact solution

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

This IVP is stiff when $Re(\lambda) \ll 0$. In this case, the solution behaves like $e^{\lambda t}$, i.e., like the solution of the linear problem $y' = \lambda y$. Suppose that this problem is solved by the backward Euler method.

1.1. Find the maximum step size h for which functional iteration (5.7.2) converges when $\lambda = -10^4$.

1.2. Show that Newton's iteration converges for much larger step sizes.

Bibliography

[1] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[2] M. Berzins and R.M. Furzeland. A user's manual for sprint - a versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 - algebraic and ordinary differential equations. Technical report, Thornton Research Centre, Shell Research Ltd., Amsterdam, 1993.

[3] W.E. Boyce and R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley and Sons, New York, third edition, 1977.

[4] K.E. Brenan, S.L Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* North Holland, New York, 1989.

[5] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh. Vode: a variable coefficient ode solver. *SIAM J. Sci. Stat. Comput.*, 10:1039–1051, 1989.

[6] R.L. Burden and J.D. Faires. *Numerical Analysis.* PWS-Kent, Boston, fifth edition, 1993.

[7] G.D. Byrne and A.C. Hindmarsh. A polyalgorithm for the numerical solution of ordinary differential equations. *ACM Trans. Math. Software*, 1:71–96, 1975.

[8] G. Dahlquist. A special stability problem for some linear multistep methods. *BIT*, 3:27–43, 1963.

[9] J.R. Dormand and P.J. Prince. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6:19–26, 1980.

[10] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice Hall, Englewood Cliffs, 1971.

[11] R.D. Grigorieff and J.Schroll. Über A(*alpha*)-stable verfahren hoher konsistenzordnung. *Computing*, 20:343–350, 1978.

[12] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, Berlin, second edition, 1993.

[13] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems.* Springer-Verlag, Berlin, 1991.

[14] P. Henrici. *Discrete Variable Mehtods in Ordinary Differential Equations.* John Wiley and Sons, New York, 1962.

[15] A. Nordsieck. On numerical integration of ordinary differential equations. *Maths. Comp.*, 16:22–49, 1962.

[16] L.F. Shampine and M.K. Gordon. *Computer Solution of Ordinary Differential Equations.* W.H. Freeman, San Francisco, 1975.

[17] G. Wanner, E Hairer, and S.P. Norsett. Order stars and stability theorems. *BIT*, 18:475–489, 1978.

[18] O. Widlund. A note on unconditionally stable linear multistep methods. *BIT*, 7:65–70, 1967.

Part 3: Two-Point Boundary Value Problems

Chapter 6

Fundamental Problems and Methods

6.1 Problems to be Solved

Several problems arising in science and engineering are modeled by differential equations that involve conditions that are specified at more than one point. Some examples follow.

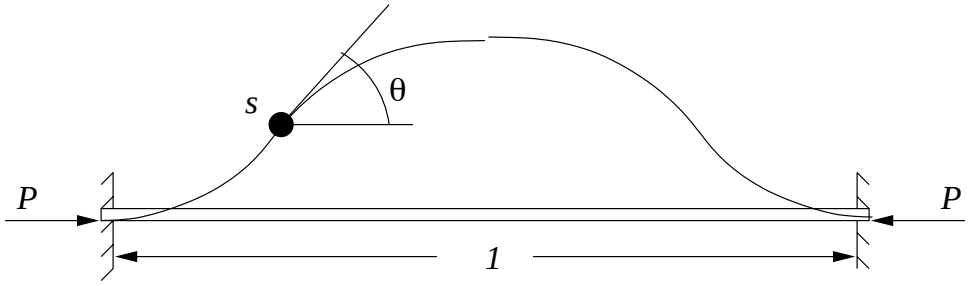

Figure 6.1.1: Deformation of an elastica.

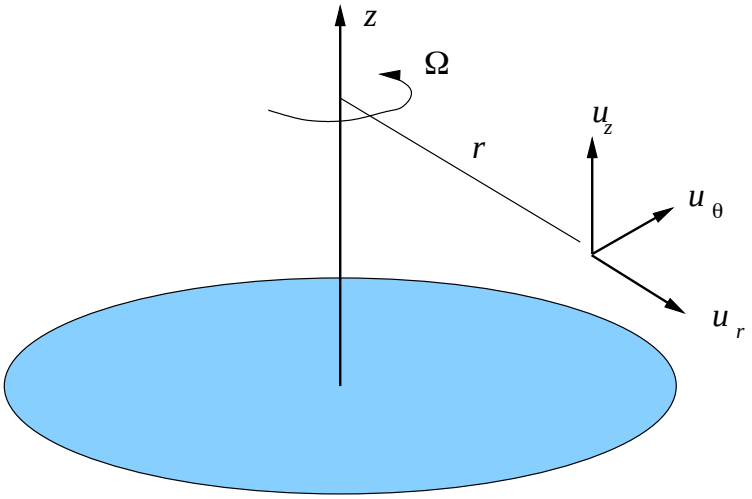

Figure 6.1.2: Swirling flow above a rotating disk.

1. *Deformation of an Elastica.* The transverse deformation of a thin elastic inextensional rod subjected to an axial loading and clamped at its ends is governed by the differential system

$$\frac{d^2\theta}{ds^2} + P\sin\theta = 0, \qquad 0 < s < 1,$$

$$\theta(0) = \theta(1) = 0.$$

As shown in Figure 6.1.1, the rod has unit length, the magnitude of the loading is P, and θ is the angle that the deformed rod makes with the initial undeformed axis. This classical second-order nonlinear two-point boundary value problem is called the *elastica problem.* One solution is $\theta = 0$. This solution, however, becomes unstable as P increases and the rod bends into a deformed shape as shown in Figure 6.1.1. Hence, this boundary value problem is also a differential eigenvalue problem that consists of determining *theta* and the critical load P for deformed shapes to exist. Once θ has been determined, the Cartesian coordinates of a deformed point on the rod can be determined as the solution of the initial value problems

$$\frac{dx}{ds} = \cos\theta, \qquad \frac{dy}{ds} = \sin\theta$$

$$x(0) = y(0) = 0.$$

2. *Swirling Flow.* The swirling flow of a viscous incompressible fluid over a disk spinning with speed Ω (Figure 6.1.2) can be analyzed by solving the nonlinear two-point boundary value problem

$$\frac{d^3f}{dx^3} + 2f\frac{d^2f}{dx^2} - (\frac{df}{dx})^2 + g^2 = \gamma^2, \qquad \frac{d^2g}{dx^2} + 2f\frac{dg}{dx} - 2\frac{df}{dx}g = 0, \qquad 0 < x < \infty,$$

$$f(0) = 0, \qquad \frac{df(0)}{dx} = 0, \qquad g(0) = 1$$

$$\lim_{x\to\infty}\frac{df(x)}{dx} = 0, \qquad \lim_{x\to\infty} g(x) = \gamma,$$

where γ is the Rosby number. The dimensionless variable x is related to the axial coordinate z (Figure 6.1.2) by

$$x = z\sqrt{\frac{\Omega}{\nu}}$$

where ν is the kinematic viscosity. The radial, tangential, and axial components of the velocity vector, respectively, are obtained from the functions $f(x)$ and $g(x)$ as

$$u_r = \Omega r \frac{df(x)}{dx}, \qquad u_\theta = \Omega r g(x), \qquad u_z = -2f(x)\sqrt{\Omega\nu}.$$

This problem involves a boundary value problem for a pair of second-order nonlinear ODEs. An interesting feature of this problem is that one of the "boundaries" is at infinity.

As indicated in these two examples, boundary value problems (BVPs) have several forms. The two that will be most important to us are:

1. A vector system of first-order equations

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), \tag{6.1.1}$$

where $\mathbf{y}' = d\mathbf{y}/dx$ and $\mathbf{y}$ and $\mathbf{f}$ are m-dimensional vectors. (We have changed the independent variable from t to x since it frequently corresponds to a spatial position rather than time.)

2. A scalar m *th*-order differential equation

$$u^{(m)} = g(x, u, u', ..., u^{(m-1)}). \tag{6.1.2}$$

Naturally, the higher-order scalar problem can be reduced to a first-order vector system (as described in Section 1.1); however, it may sometimes be convenient to work with the higher-order scalar problem (6.1.2).

Focusing on the vector system (6.1.1) for the moment, if

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{A}(x)\mathbf{y} + \mathbf{b}(x)$$

the ODE is *linear*, otherwise it is *nonlinear.*

If (6.1.1) is to be solved on $a < x < b$ then m conditions are needed to uniquely determine the solution of the BVP. These may be of the general form

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0} \tag{6.1.3a}$$

where $\mathbf{g}$ has dimension m. For the most part, we will consider simpler boundary conditions having the form

$$\mathbf{g}_L(\mathbf{y}(a)) = \mathbf{0}, \qquad \mathbf{g}_R(\mathbf{y}(b)) = \mathbf{0} \tag{6.1.3b}$$

where $\mathbf{g}_L$ has dimension l and $\mathbf{g}_R$ has dimension $r = m - l$. Conditions of the form (6.1.3b) are called *separated* while the more general form (6.1.3a) are *unseparated*.

Linear versions of (6.1.3a) and (6.1.3b) have the forms

$$\mathbf{L}\mathbf{y}(a) + \mathbf{R}\mathbf{y}(b) = \mathbf{c} \tag{6.1.4a}$$

and

$$\mathbf{L}_L\mathbf{y}(a) = \mathbf{c}_L, \qquad \mathbf{R}_R\mathbf{y}(b) = \mathbf{c}_R. \tag{6.1.4b}$$

The matrices $\mathbf{L}$ and $\mathbf{R}$ are of dimension $m \times m$ and the vector $\mathbf{c}$ is m-dimensional. For the separated conditions (6.1.4b), $\mathbf{L}_L$ is $l \times m$-dimensional, $\mathbf{R}_R$ is $r \times m$-dimensional, $\mathbf{c}_L$ is l-dimensional, and $\mathbf{c}_R$ is r-dimensional.

There are three standard numerical approaches to solving two-point boundary value problems:

1. *Shooting.* An appropriate IVP is defined and solved by initial value techniques and software. The IVP is defined so that solutions iteratively converge to the solution of the original BVP.

2. *Finite Differences.* A mesh is introduced on $[a, b]$ and derivatives in the ODE are replaced by finite-differences relative to the mesh. This leads to a linear or nonlinear algebraic problem which may be solved to produce a discrete approximation of the solution of the BVP.

3. *Projections.* The solution of the BVP is approximated by simpler functions, e.g., piecewise polynomials, and the differential equations and boundary conditions are satisfied approximately. Collocation or finite element techniques often furnish these approximations.

In the next three Sections we illustrate the basic ideas of these methods by using simple BVPs.

6.2 Introduction to Shooting

Let us consider a second-order nonlinear two-point BVP

$$u''(x) = f(x, u, u'), \qquad a < x < b, \qquad u(a) = A, \qquad u(b) = B. \tag{6.2.1}$$

Writing the ODE as a first-order system, let us also consider the related IVP

$$y_1' = y_2, \qquad y_1(a; \alpha) = A, \tag{6.2.2a}$$

$$y_2' = f(x, y_1, y_2), \qquad y_2(a; \alpha) = \alpha. \tag{6.2.2b}$$

In what follows, we'll need to emphasize the dependence of the solution on the parameter α appearing in the initial conditions, so we'll write the solution of (6.2.2) as $y_k(x; \alpha)$, $k = 1, 2$.

Solutions of (6.2.2) satisfy the original differential equation and the initial condition at $x = a$ but fail to satisfy the terminal condition at $x = b$. Thus, shooting consists of repeatedly solving (6.2.2) for different choices of α until the terminal condition

$$y_1(b; \alpha) = B \tag{6.2.3}$$

is also satisfied. Regarding (6.2.3) as a (nonlinear) function of α, convergence to the solution of the BVP can be enhanced by using an iterative strategy for nonlinear algebraic equations. Secant and Newton iteration are two possible procedures. Let us illustrate the simpler secant method first.

1. Solve the IVP (6.2.2) for two choices of α, say α_0 and α_1. The corresponding solutions, $y_1(x; \alpha_0)$ and $y_1(x; \alpha_1)$, may appear as illustrated in Figure 6.2.1. The various choices of α alter the initial slope $y_1'(a; \alpha) = y_2(a; \alpha)$. Regarding the solution $y_1(x; \alpha)$ as the trajectory of a projectile fired from a cannon at $x = a$, $y_1(a; \alpha) = A$, the problem is to alter the initial angle of the cannon so that the projectile hits a target at at $x = b$, $y_1(a; \alpha) = B$; hence, the name shooting.

2. Assuming that $y_1(b; \alpha)$ is locally a linear function of α, we use the two values $y_1(b, \alpha_0)$ and $y_1(b, \alpha_1)$ to compute the next value α_2 in the sequence; thus, α_2 is the

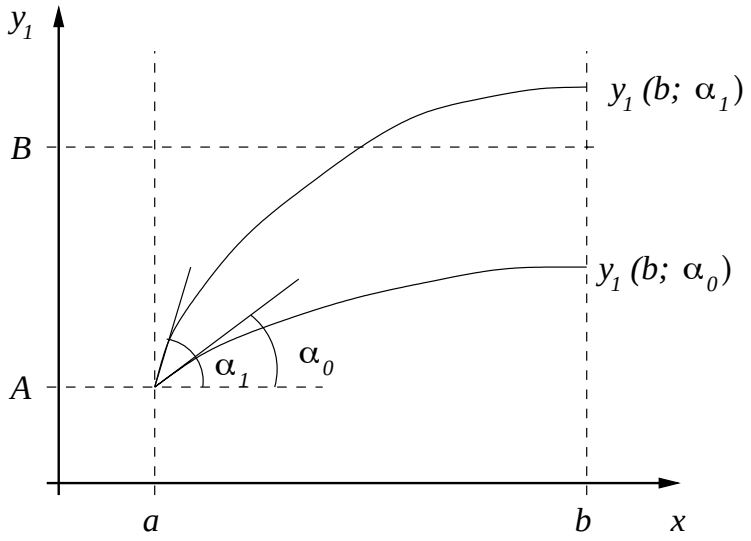

Figure 6.2.1: Solutions $y_1(x, \alpha_0)$ and $y_1(x, \alpha_1)$ of the IVP (6.2.2).

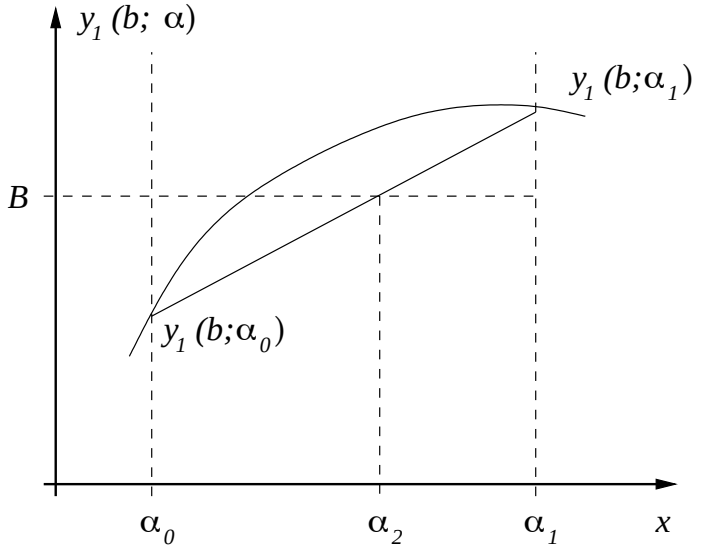

Figure 6.2.2: Secant method of using two guesses α_0 and α_1 to select another guess α_2 such that $y_1(b; \alpha_2) \approx B$.

solution of (Figure 6.2.2)

$$\frac{y_1(b;\alpha_1) - B}{\alpha_1 - \alpha_2} = \frac{y_1(b;\alpha_1) - y_1(b;\alpha_0)}{\alpha_1 - \alpha_0}.$$

Solving for α_2 yields

$$\alpha_2 = \alpha_1 - (\alpha_1 - \alpha_0)\frac{y_1(b;\alpha_1) - B}{y_1(b;\alpha_1) - y_1(b;\alpha_0)}.$$

This can be repeated to yield the general relation

$$\alpha_{\nu+1} = \alpha_\nu - (\alpha_\nu - \alpha_{\nu-1})\frac{y_1(b;\alpha_\nu) - B}{y_1(b;\alpha_\nu) - y_1(b;\alpha_{\nu-1})}, \qquad \nu = 1, 2, \ldots . \tag{6.2.4}$$

The iteration may be terminated when, e.g.,

$$\frac{y_1(b;\alpha_\nu) - B}{B} \leq \epsilon$$

for a prescribed value of ϵ. (Other termination criteria should be used when $B = 0$.)

Remark 1. If the ODE is linear then $y_1(b;\alpha)$ is a linear function of α and $y_1(x;\alpha_2)$ is the exact solution (neglecting round off errors) of the BVP (6.2.1).

The nonlinear equation (6.2.3) can also be solved by Newton's method. If, for example, α_ν is a (sufficiently close) guess to the solution of (6.2.3) then the next guess may be generated as

$$\alpha_{\nu+1} = \alpha_\nu - \frac{y_1(b;\alpha_\nu) - B}{\frac{\partial y_1(b;\alpha_\nu)}{\partial \alpha}}. \tag{6.2.5}$$

An expression for $\partial y_1(b;\alpha_\nu)/\partial\alpha$ may be obtained by differentiating the IVP (6.2.2) with respect to α to obtain

$$(\frac{\partial y_1}{\partial \alpha})' = \frac{\partial y_2}{\partial \alpha}, \qquad \frac{\partial y_1(a;\alpha)}{\partial \alpha} = 0, \tag{6.2.6a}$$

$$(\frac{\partial y_2}{\partial \alpha})' = \frac{\partial f(x, y_1, y_2)}{\partial y_1}\frac{\partial y_1}{\partial \alpha} + \frac{\partial f(x, y_1, y_2)}{\partial y_2}\frac{\partial y_2}{\partial \alpha}, \qquad \frac{\partial y_2(a;\alpha)}{\partial \alpha} = 1. \tag{6.2.6b}$$

These equations are linear in the partial derivatives $\partial y_1/\partial\alpha$ and $\partial y_2/\partial\alpha$.

An algorithm for performing shooting with Newton's method is shown in Figure 6.2.3. In order to simplify the notation, let

$$z_1 = \frac{\partial y_1}{\partial \alpha}, \qquad z_2 = \frac{\partial y_2}{\partial \alpha}. \tag{6.2.7}$$

In order to solve the IVP, functions to evaluate $\partial f/\partial y_1$ and $\partial f/\partial y_2$ must be available In contrast, the secant method only requires knowledge of f. In fact, the secant method (6.2.4) can be viewed as an approximation of Newton's method (6.2.5) with backward

procedure newton;
  begin
    Select an initial guess α_0;
    $\nu := 0$;
    repeat
      Solve the IVP for $x \in (a, b]$
        $y_1' = y_2,\ y_1(a;\alpha_\nu) = A,$
        $y_2' = f(x, y_1, y_2),\ y_2(a;\alpha_\nu) = \alpha_\nu,$
        $z_1' = z_2,\ z_1(a;\alpha_\nu) = 0,$
        $z_2 = f_{y_1}(x, y_1, y_2)z_1 + f_{y_2}(x, y_1, y_2)z_2,\ z_2(a;\alpha_\nu) = 1;$
      if not converged **then**
        begin
          $\alpha_{\nu+1} := \alpha_\nu - \frac{y_1(b;\alpha_\nu)-B}{z_1(b,\alpha_\nu)}$;
          $\nu := \nu + 1$
        end
    until converged;
  end;

Figure 6.2.3: The shooting method for solving (6.2.1) with Newton iteration.

differences replacing $\partial y_1(b, \alpha_\nu)/\partial\alpha$. For second-order BVPs, Newton's method requires the solution of a four-dimensional IVP while the secant method only requires a two-dimensional IVP.

Convergence of Newton's method is generally second-order (quadratic), i.e.,

$$|\alpha_{\nu+1} - \alpha_\infty| \le C|\alpha_\nu - \alpha_\infty|^2, \qquad \nu \to \infty,$$

where α_∞ is the value of α that satisfies the terminal condition (6.2.3). Convergence of the secant method is slightly slower, typically

$$|\alpha_{\nu+1} - \alpha_\infty| \le C|\alpha_\nu - \alpha_\infty|^{1.5}, \qquad \nu \to \infty,$$

Thus, the secant method would generally be preferred to Newton's method. This, however, may not be the case with higher-dimensional BVPs.

Example 6.2.1. Consider the solution of the clamped elastica problem

$$\theta'' + P\sin\theta = 0, \qquad \theta(0) = \theta(1/2) = 0.$$

by shooting methods using Newton iteration. (Symmetry considerations have been used to cut the domain of the problem illustrated in Figure 6.1.1 in half.)

Letting

$$y_1 = \theta, \qquad y_2 = \theta',$$

we introduce the IVP

$$y_1' = y_2, \qquad y_1(0;\alpha) = 0,$$

$$y_2' = -P \sin y_1, \qquad y_2(0;\alpha) = \alpha.$$

Differentiating this system with respect to α yields

$$z_1' = z_2, \qquad z_1(0;\alpha) = 0,$$

$$z_2' = -P z_1 \cos y_1, \qquad z_2(0;\alpha) = 1,$$

where z_k, $k = 1, 2$, satisfies (6.2.7). Iterates are computed by the relation

$$\alpha_{\nu+1} = \alpha_\nu - \frac{y_1(1/2;\alpha_\nu)}{z_1(1/2;\alpha_\nu)}.$$

Using a convergence test of

$$|y_1(1/2;\alpha_\nu)| \le 10^{-9}$$

we found that Newton's method converged in five iterations when $P = 40$ and $\alpha_0 = 0.1$.

6.3 Introduction to Finite Difference Methods

We'll again use the second-order scalar nonlinear two-point boundary value problem

$$y''(x) = f(x, y, y'), \qquad a < x < b, \qquad y(a) = A, \qquad y(b) = B, \tag{6.3.1}$$

to describe the essential details of finite difference methods.

To begin, we divide the domain $a \le x \le b$ into N uniform subintervals of width

$$h = \frac{b-a}{N}, \tag{6.3.2a}$$

as shown in Figure 6.3.1. Restriction to uniform subintervals is not essential, but is introduced here for simplicity. We also let

$$x_i = a + ih, \qquad i = 0, 1, \ldots, N. \tag{6.3.2b}$$

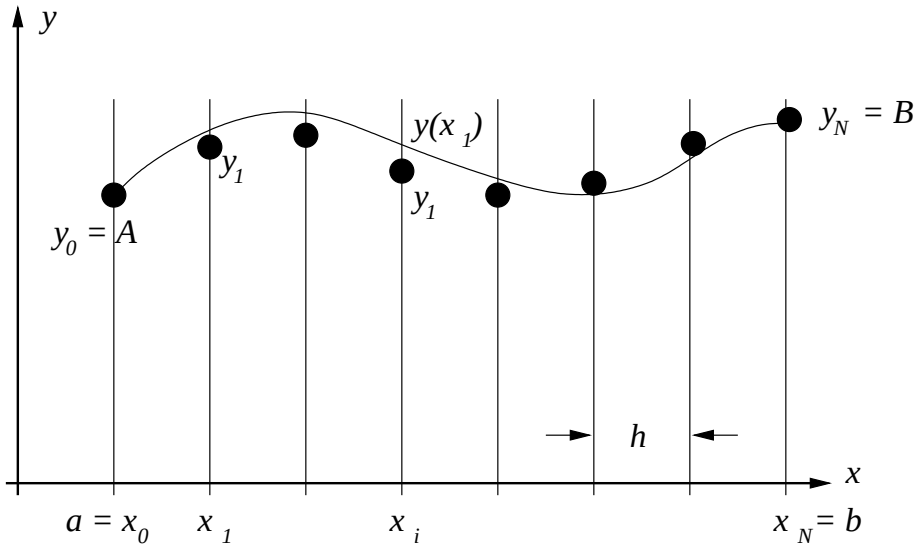

Figure 6.3.1: Domain, discretization, and notation used for finite difference solutions of (6.3.1).

In solving the BVP (6.3.1) by finite differences, all derivatives are replaced by finite difference approximations. These can be constructed from interpolating polynomials, but we'll illustrate a different approach using Taylor's series expansions of the solution $y(x)$. Thus, consider

$$y(x) = y(x_i) + (x - x_i)y'(x_i) + \frac{(x - x_i)^2}{2!}y''(x_i) + \ldots + \frac{(x - x_i)^k}{k!}y^{(k)}(\xi) \qquad (6.3.3)$$

where ξ is between x_i and x. Specifically, choosing $x = x_{i+1} = x_i + h$ yields

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) + \frac{h^3}{6}y'''(x_i) + \ldots + \frac{h^k}{k!}y^{(k)}(\xi_{i+1}). \qquad (6.3.4a)$$

Similarly, selecting $x = x_{i-1} = x_i - h$ produces

$$y(x_{i-1}) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) - \frac{h^3}{6}y'''(x_i) + \ldots + \frac{(-h)^k}{k!}y^{(k)}(\xi_{i-1}). \qquad (6.3.4b)$$

Setting $k = 2$ in (6.3.4a) and solving for $y'(x_i)$ yields

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_i)}{h} - \frac{h}{2}y''(\xi_{i+1}). \qquad (6.3.5a)$$

Finite difference approximations are obtained by neglecting the error term of the Taylor's series; thus, the first forward finite difference approximation of $y'(x_i)$ is

$$y'_i = \frac{y_{i+1} - y_i}{h} \qquad (6.3.5b)$$

and the local discretization error of this approximation is

$$\tau_i = -\frac{h}{2}y''(\xi_{i+1}). \tag{6.3.5c}$$

Subscripts on y denote finite difference approximations; hence, y_i denotes an approximation of $y(x_i)$, *etc.*

In a similar manner, the first backward difference approximation of $y'(x_i)$ is obtained by setting $k = 2$ in (6.3.4b)

$$y'_i = \frac{y_i - y_{i-1}}{h}, \qquad \tau_i = \frac{h}{2}y''(\xi_{i-1}). \tag{6.3.6}$$

Notice, however, that a higher-order and symmetric difference approximation can be obtained by subtracting (6.3.4b) from (6.3.4a) and setting $k = 3$ to get

$$y(x_{i+1}) - y(x_{i-1}) = 2hy'(x_i) + \frac{h^3}{3}y'''(\xi_i).$$

Solving for $y'(x_i)$

$$y'_i = \frac{y_{i+1} - y_{i-1}}{2h}, \qquad \tau_i = -\frac{h^2}{6}y'''(\xi_i). \tag{6.3.7}$$

The difference formula (6.3.7) is called the *first central difference* approximation of $y'(x_i)$. In Chapter 5, we found that this approximation led to the leap frog scheme, which had poor stability characteristics. Here, with second-order ODEs, central differences will generally be preferred to either forward or backward differences because of their higher-order local discretization errors.

Remark 1. The higher-order accuracy of (6.3.7) relative to (6.3.5) or (6.3.6) only occurs on a uniform mesh. With nonuniform spacing the second derivative terms in (6.3.4a) and (6.3.4b) would not cancel upon subtraction.

A central difference approximation of the second derivative $y''(x_i)$ is obtained by adding (6.3.4a) and (6.3.4b) while setting $k = 4$ to obtain

$$y''_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \qquad \tau_i = -\frac{h^2}{12}y^{iv}(\xi_i). \tag{6.3.8}$$

No further approximations are needed to solve (6.3.1) by finite differences; however, we note that approximations of higher derivatives are obtained by using Taylor's series

at more points. For example, consider evaluating the Taylor's series (6.3.3) at $x = x_{i+2}$ and x_{i-2} to obtain

$$y(x_{i+2}) = y(x_i) + 2hy'(x_i) + 2h^2y''(x_i) + \frac{4h^3}{3}y'''(x_i) + \frac{2h^4}{3}y^{iv}(x_i) + \ldots \tag{6.3.9a}$$

and

$$y(x_{i-2}) = y(x_i) - 2hy'(x_i) + 2h^2y''(x_i) - \frac{4h^3}{3}y'''(x_i) + \frac{2h^4}{3}y^{iv}(x_i) + \ldots . \tag{6.3.9b}$$

Subtracting (6.3.9b) from (6.3.9a) yields

$$y(x_{i+2}) - y(x_{i-2}) = 4hy'(x_i) + \frac{8h^3}{3}y'''(x_i) + O(h^5).$$

A similar subtraction of (6.3.4b) from (6.3.4a) yields

$$y(x_{i+1}) - y(x_{i-1}) = 2hy'(x_i) + \frac{h^3}{3}y'''(x_i) + O(h^5).$$

Elimination of the first derivative term yields a central difference approximation of the third derivative as

$$y_i''' = \frac{y_{i+2} - 2y_{i+1} + 2y_{i-1} - y_{i-2}}{2h^3}. \tag{6.3.10}$$

The local discretization error $\tau_i = O(h^2)$.

Similar combinations of (6.3.4) and (6.3.9) yield an $O(h^2)$ central difference approximation of the fourth derivative as

$$y_i^{iv} = \frac{y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}}{h^4}. \tag{6.3.11}$$

Now let us return to the task of solving (6.3.1) by finite difference approximations. We'll try central-difference approximations because of their higher order. Thus, evaluating (6.3.1) at $x = x_i$ and replacing derivatives by central differences using (6.3.7a) and (6.3.8a), we obtain

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}) \tag{6.3.12a}$$

Writing (6.3.12a) at each interior mesh point $i = 1, 2, \ldots, N-1$, and using the two boundary conditions

$$y_0 = A, \qquad y_N = B \tag{6.3.12b}$$

gives a system of $N+1$ nonlinear algebraic equations in the $N+1$ unknowns y_i, $i = 0, 1, \ldots, N$. This system is too complex for an introductory example, so let us confine our attention to linear problems with

$$f(x, y, y') = -p(x)y' - q(x)y + r(x). \tag{6.3.13}$$

In this case, the approximation (6.3.12a) becomes

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = r_i, \qquad i = 1, 2, \ldots, N-1, \tag{6.3.14}$$

where $p_i = p(x_i)$, *etc.* Referring to this as the central-difference approximation of the ODE we define the local discretization error as follows.

Definition 6.3.1. *Consider an ODE in the form $\mathcal{L}y(x) = 0$ and let $\mathcal{L}_h y$ be a difference approximation of it, with $\mathcal{L}$ and $\mathcal{L}_h$ being differential and difference operators. The local discretization error or the local truncation error at $x = x_i$ is*

$$\tau_i = \mathcal{L}_h y(x_i). \tag{6.3.15}$$

Example 6.3.1. The differential and difference operators for the linear ODE (6.3.1, 6.3.13) satisfy

$$\mathcal{L}y(x) = y'' + p(x)y' + q(x)y - r(x) = 0$$

and

$$\mathcal{L}_h y(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} + p(x_i)\frac{y(x_{i+1}) - y(_{i-1})}{2h} + q(x_i)y(x_i) - r(x_i).$$

Using (6.3.7) and (6.3.8), we find

$$\tau_i = y''(x_i) + \frac{h^2}{12}y^{iv}(\xi_i) + p(x_i)[y'(x_i) + \frac{h^2}{6}y'''(\eta_i)] + q(x_i)y(x_i) - r(x_i).$$

Using the differential equation

$$\tau_i = \frac{h^2}{12}y^{iv}(\xi_i) + p(x_i)\frac{h^2}{6}y'''(\eta_i).$$

Thus, as we might have expected, the local discretization of the central difference approximation of (6.3.1, 6.3.13) is $O(h^2)$.

The algebraic system (6.3.12b, 6.3.14) is linear for the linear BVP (6.3.1, 6.3.12b, 6.3.13) and may be solved by, e.g., Gaussian elimination. Towards this end, let us write (6.3.13) in the form

$$b_i y_{i-1} + a_i y_i + c_i y_{i+1} = h^2 r_i, \qquad i = 1, 2, \ldots, N-1, \tag{6.3.16a}$$

where

$$b_i = 1 - \frac{h}{2} p_i, \qquad a_i = -2 + h^2 q_i, \qquad c_i = 1 + \frac{h}{2} p_i. \tag{6.3.16b}$$

The boundary conditions (6.3.12b) may be used in conjunction with (6.3.16a) to create a system of dimension $N+1$ or used to explicitly eliminate y_0 and y_N as unknowns from (6.3.16a). The latter approach is preferred for simple problems like this one. Thus, using (6.3.12b) with (6.3.16a) when $i = 1$, we find

$$a_1 y_1 + c_1 y_2 = h^2 r_1 - b_1 A. \tag{6.3.17a}$$

Similarly, using (6.3.12b) with (6.3.16a) when $i = N-1$ yields

$$b_{N-1} y_{N-2} + a_{N-1} y_{N-1} = h^2 r_{N-1} - c_{N-1} B. \tag{6.3.17b}$$

Grouping the $N-1$ equations, (6.3.17a), (6.3.17b), and (6.3.16a), $i = 2, 3, \ldots, N-2$, yields

$$\mathbf{A}\mathbf{y} = \mathbf{f}, \tag{6.3.18a}$$

where

$$\mathbf{A} = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{N-2} & a_{N-2} & c_{N-2} \\ & & & b_{N-1} & a_{N-1} \end{bmatrix}, \tag{6.3.18b}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} h^2 r_1 - b_1 A \\ h^2 r_2 \\ \vdots \\ h^2 r_{N-2} \\ h^2 r_{N-1} - c_{N-1} B \end{bmatrix}. \tag{6.3.18c}$$

The linear algebraic problem (6.3.18) requires the solution of a tridiagonal system to determine the $N-1$ unknowns y_i, $i = 1, 2, \ldots, N-1$. The basic solution strategy is Gaussian elimination. Pivoting is frequently unnecessary. As seen from (6.3.16b), $\mathbf{A}$ will be diagonally dominant unless $|p(x)|$ is large relative to $|q(x)|$ or h is too small. Pivoting and other special treatment may be necessary in these exceptional situations. Let us proceed without pivoting and write $\mathbf{A}$ in the slightly more general form

$$\mathbf{A} = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_{n-1} \\ & & & b_n & a_n \end{bmatrix} \tag{6.3.19a}$$

where, in our case, $n = N-1$. We factor $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{LU} \tag{6.3.19b}$$

where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. Having performed this factorization, we write (6.3.18a) as

$$\mathbf{LUy} = \mathbf{f},$$

let

$$\mathbf{Uy} = \mathbf{z}, \tag{6.3.19c}$$

and solve

$$\mathbf{Lz} = \mathbf{f}. \tag{6.3.19d}$$

Since $\mathbf{L}$ is lower triangular, (6.3.19d) may be solved for $\mathbf{z}$ by forward substitution. Knowing $\mathbf{z}$, we determine $\mathbf{y}$ by solving (6.3.19c) by backward substitution. All that remains is the determination of $\mathbf{L}$ and $\mathbf{U}$. Let us hypothesize that they have the following bidiagonal forms

$$\mathbf{L} = \begin{bmatrix} 1 & & & & \\ \beta_2 & 1 & & & \\ & \beta_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \beta_n & 1 \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} \alpha_1 & \gamma_1 & & & \\ & \alpha_2 & \gamma_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & \gamma_{n-1} \\ & & & & \alpha_n \end{bmatrix}. \tag{6.3.20}$$

Using (6.3.20) with (6.3.19a,b) we find

$$\alpha_1 = a_1, \tag{6.3.21a}$$

$$\beta_i = b_i/\alpha_{i-1}, \tag{6.3.21b}$$

$$\alpha_i = a_i - \beta_i\gamma_{i-1}, \qquad i = 2, 3, \ldots, n, \tag{6.3.21c}$$

$$\gamma_i = c_i, \qquad i = 1, 2, \ldots, n-1. \tag{6.3.21d}$$

Using (6.3.20a) and (6.3.19d) we find the forward substitution step to be

$$z_1 = f_1, \tag{6.3.22a}$$

$$z_i = f_i - \beta_i z_{i-1}, \qquad i = 2, 3, \ldots, n, \tag{6.3.22b}$$

Similarly, using (6.3.20b) and (6.3.19c), the backward substitution step is

$$y_n = z_n/\alpha_n, \tag{6.3.23a}$$

$$y_i = (z_i - \gamma_i y_{i+1})/\alpha_i, \qquad i = n-1, n-2, \ldots, 1. \tag{6.3.23b}$$

The solution procedure defined by (6.3.19 - 6.3.23) is the basis of the famous tridiagonal algorithm. We state it as a pseudo-PASCAL algorithm in Figure 6.3.2. The version implemented in Figure 6.3.2 overwrites a_i, b_i, and c_i with α_i, β_i, and γ_i to reduce storage. By counting we see that the algorithm requires approximately $5n$ multiplications or divisions and $3n$ additions and subtractions. The work required to factor a full matrix by Gaussian elimination is approximately $n^3/3$. Thus, the ratio of the work to factor a tridiagonal matrix to that of a full matrix is approximately $15/n^2$. This is a significant savings even for small matrices and one should never use a Gaussian elimination procedure for full matrices to solve a tridiagonal system

Example 6.3.2. Evidence from the Taylor's series expansion would suggest that the global error of the finite difference solution of the BVP (6.3.1, 6.3.12b, 6.3.13) has an

```
Procedure tridi(n : integer; var a, b, c, f, y : array [1..n] of real);
  begin

    { Factorization }
    for i = 2 to n do
      begin
        b_i := b_i/a_{i-1};
        a_i := a_i − b_i c_{i-1}
      end;

    { Forward substitution }
    y_1 := f_1;
    for i = 2 to n do y_i := f_i − b_i y_{i-1};

    { Backward substitution }
    y_n := y_n/a_n;
    for i = n − 1 downto 1 do y_i := (y_i − c_i y_{i+1})/a_i
  end;
```

Figure 6.3.2: Tridiagonal algorithm

expansion in even powers of h beginning with $O(h^2)$ terms. Let's assume that this is the case. Then, Richardson's extrapolation can be used to both estimate the global error and to improve the solution. To this end, we calculate two solutions using different step sizes of, e.g., h and $h/2$. In order to emphasize the dependence of the discrete solution on step size, let y_i^h denote the finite difference solution y_i at $x_i = a + ih$ obtained with step size h. With the assumed error dependence we have

$$y(a + ih) - y_i^h = Ch^2 + O(h^4)$$

and

$$y(a + ih) - y_{2i}^{h/2} = C(\frac{h}{2})^2 + O(h^4).$$

The variable $y_{2i}^{h/2}$ is the finite difference solution at $a + 2i(h/2) = a + ih$.

Subtracting the two error equations to eliminate the exact solution yields

$$Ch^2 = \frac{4}{3}[y_{2i}^{h/2} - y_i^h] + O(h^4).$$

Using this result, we estimate the error in the finer grid solution as

$$y(a+ih) - y_{2i}^{h/2} \approx \frac{y_{2i}^{h/2} - y_i^h}{3}.$$

Furthermore,

$$\hat{y}_i^{h/2} = y_{2i}^{h/2} + \frac{y_{2i}^{h/2} - y_i^h}{3}, \qquad i = 1, 2, \ldots, N-1,$$

is an $O(h^4)$ approximation of the solution.

Let us apply Richardson's extrapolation to the simple BVP

$$y'' = y, \qquad y(0) = 0, \qquad y(1) = 1.$$

This problem has the form of (6.3.13) with $p(x) = r(x) = 0$ and $q(x) = -1$. Thus, the elements of the tridiagonal system (6.3.18) are

$$a_i = -(2 + h^2), \qquad i = 1, 2, \ldots, N-1,$$

$$b_i = 1, \qquad i = 2, 3, \ldots, N-1, \qquad c_i = 1, \qquad i = 1, 2, \ldots, N-2.$$

Central-difference solutions with $h = 1/10, 1/20$, the solution by Richardson's extrapolation, and the exact solution

$$y(x) = \frac{\sinh x}{\sinh 1}$$

are shown in Table 6.3.1. Using the error at $x = 0.5$ as a measure of accuracy, we have

$$|y(0.5) - y_5^{0.1}| = 4.26 \times 10^{-5},$$

$$|y(0.5) - y_{10}^{0.05}| = 1.04 \times 10^{-5},$$

$$\frac{|y_{10}^{0.5} - y_5^{0.1}|}{3} = 1.07 \times 10^{-5},$$

$$|y(0.5) - \hat{y}_5^{0.05}| = 3.3 \times 10^{-7},$$

These results indicate that

1. the global error of the centered finite difference solution is approximately $O(h^2)$ since decreasing h by one-half quarters the error and

2. Richardson's extrapolation furnishes a good approximation of the error while also improving accuracy.

i	x_i	$y(x_i)$	y_i^h	$y_{2i}^{h/2}$	$\hat{y}_i^h/2$
0	0.00	0.0	0.0	0.0	0.0
1	0.05	0.04256364		0.04256498	
2	0.10	0.08523370	0.08524467	0.08523638	0.08523361
3	0.15	0.12811690		0.12812087	
4	0.20	0.17132045	0.17134180	0.17132566	0.17132029
5	0.25	0.21495240		0.21495877	
6	0.30	0.25912184	0.25915234	0.25912928	0.25912159
7	0.35	0.30393920		0.30394761	
8	0.40	0.34951658	0.34955441	0.34952582	0.34951629
9	0.45	0.39596749		0.39597785	
10	0.50	0.44340942	0.44345203	0.44341982	0.44340909
11	0.55	0.49195965		0.49197036	
12	0.60	0.54174004	0.54178417	0.54175082	0.54173970
13	0.65	0.59287506		0.59288567	
14	0.70	0.64549258	0.64553415	0.64550275	0.64549229
15	0.75	0.69972415		0.69973360	
16	0.80	0.75570543	0.75573949	0.75571378	0.75570522
17	0.85	0.81357636		0.81358325	
18	0.90	0.87348163	0.87350223	0.87348670	0.87348153
19	0.95	0.93557107		0.93557388	
20	1.00	1.0	1.0	1.0	1.0

Table 6.3.1: Solution of Example 6.3.2 using central difference approximations and Richardson's extrapolation.

As a next step, let us consider a linear BVP with a prescribed Robin boundary condition, e.g.,

$$y'' + p(x)y' + q(x)y = r(x), \qquad a < x < b, \tag{6.3.24a}$$

$$y(a) = A, \tag{6.3.24b}$$

$$y'(b) + Cy(b) = B. \tag{6.3.24c}$$

As distinct from the Dirichlet conditions used in (6.3.1), $y(b)$ is now an unknown. We could approximate $y'(b)$ by backward differences and use the discrete version of the terminal condition (6.3.24c) to determine an approximation of $y(b)$; however, this has some drawbacks. If first-order backward differences (6.3.6) were used to approximate $y'(b)$

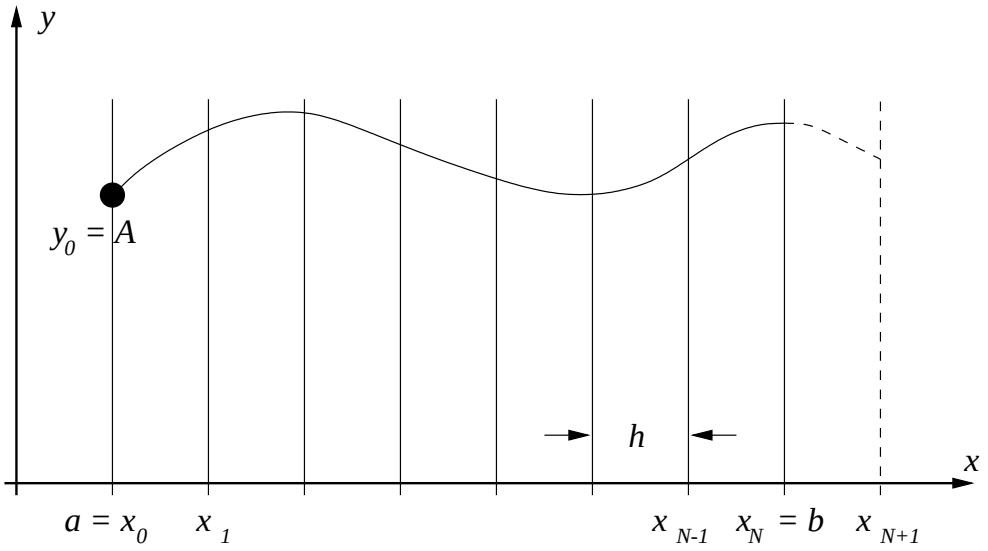

Figure 6.3.3: Domain and discretization used to approximate a Robin terminal condition.

then the boundary condition (6.3.24c) would only be accurate to $O(h)$ while the discrete approximation of the ODE (6.3.24a) is accurate to $O(h^2)$. If higher-order backward differences were used to approximate $y'(b)$ then the tridiagonal structure of the discrete system would be lost.

The usual strategy is to introduce a fictitious external point $x_{N+1} = b + h$ as shown in Figure 6.3.3. Extending the solution to this exterior point, we use central differences to approximate the terminal condition (6.3.24c) to $O(h^2)$ as

$$\frac{y_{N+1} - y_{N-1}}{2h} + Cy_N = B. \tag{6.3.25a}$$

This does little to solve the problem since we've introduced both another equation (6.3.25a) and another unknown y_{N+1}. The additional equation that we need is the central difference approximation of the ODE (6.3.24a) at $x = x_N$. Thus, using (6.3.16a) with $i = N$ we have

$$b_N y_{N-1} + a_N y_N + c_N y_{N+1} = h^2 r_N. \tag{6.3.25b}$$

Once again, It is common to eliminate y_{N+1} by combining (6.3.25a) and (6.3.25b) to obtain

$$(b_N + c_N)y_{N-1} + (a_N - 2hCc_N)y_N = h^2 r_N - 2hc_N B. \tag{6.3.25c}$$

Observing that $b_N + c_N = 2$ by use of (6.3.16b), we obtain the tridiagonal system

$$\begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{N-1} & a_{N-1} & c_{N-1} \\ & & & 2 & a_N - 2hCc_N \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} h^2 r_1 - b_1 A \\ h^2 r_2 \\ \vdots \\ h^2 r_{N-1} \\ h^2 r_N - 2hc_N B \end{bmatrix}, \qquad (6.3.26)$$

which may be solved by the tridiagonal algorithm of Figure 6.3.2.

Now let us return to the original nonlinear problem (6.3.12). Most iterative schemes for solving nonlinear algebraic equations can be used to determine the solution, but we'll illustrate the use of Newton's method, which is the most popular. To begin, let us write the finite difference system (6.3.12a) in the form

$$F_i(\mathbf{y}) = y_{i-1} - 2y_i + y_{i+1} - h^2 f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}) = 0, \qquad i = 1, 2, \ldots, N-1, \qquad (6.3.27)$$

subject to the Dirichlet boundary conditions (6.3.12b) and the definition of the vector of unknowns $\mathbf{y}$ given by (6.3.18c).

Newton's iteration involves solving

$$\mathbf{F}_{\mathbf{y}}(\mathbf{y}^{(\nu)})(\mathbf{y}^{(\nu+1)} - \mathbf{y}^{(\nu)}) = -\mathbf{F}(\mathbf{y}^{(\nu)}), \qquad \nu = 0, 1, \ldots, \qquad (6.3.28a)$$

where

$$\mathbf{F}(\mathbf{y}) = \begin{bmatrix} F_1(\mathbf{y}) \\ F_2(\mathbf{y}) \\ \vdots \\ F_{N-1}(\mathbf{y}) \end{bmatrix}, \qquad \mathbf{F}_{\mathbf{y}}(\mathbf{y}) = \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_1}{\partial y_2} & \cdots & \frac{\partial F_1}{\partial y_{N-1}} \\ \frac{\partial F_2}{\partial y_1} & \frac{\partial F_2}{\partial y_2} & \cdots & \frac{\partial F_2}{\partial y_{N-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_{N-1}}{\partial y_1} & \frac{\partial F_{N-1}}{\partial y_2} & \cdots & \frac{\partial F_{N-1}}{\partial y_{N-1}} \end{bmatrix}. \qquad (6.3.28b)$$

Differentiating (6.3.27), the Jacobian $\mathbf{F}_{\mathbf{y}}(\mathbf{y})$ is

$$\frac{\partial F_i}{\partial y_j} = \begin{cases} 1 + \frac{h}{2}\frac{\partial f}{\partial y'}, & \text{if } j = i-1 \\ -2 - h^2 \frac{\partial f}{\partial y}, & \text{if } j = i \\ 1 - \frac{h}{2}\frac{\partial f}{\partial y'}, & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases}. \qquad (6.3.29)$$

Letting

$$b_i^{(\nu)} = 1 + \frac{h}{2}\frac{\partial f}{\partial y'}(x_i, y_i^{(\nu)}, \frac{y_{i+1}^{(\nu)} - y_{i-1}^{(\nu)}}{2h}), \qquad (6.3.30a)$$

$$a_i^{(\nu)} = -2 - h^2 \frac{\partial f}{\partial y}(x_i, y_i^{(\nu)}, \frac{y_{i+1}^{(\nu)} - y_{i-1}^{(\nu)}}{2h}), \tag{6.3.30b}$$

$$c_i^{(\nu)} = 1 - \frac{h}{2}\frac{\partial f}{\partial y'}(x_i, y_i^{(\nu)}, \frac{y_{i+1}^{(\nu)} - y_{i-1}^{(\nu)}}{2h}). \tag{6.3.30c}$$

gives

$$\mathbf{F}_{\mathbf{y}}(\mathbf{y}^{(\nu)}) = \begin{bmatrix} a_1^{(\nu)} & c_1^{(\nu)} & & & \\ b_2^{(\nu)} & a_2^{(\nu)} & c_2^{(\nu)} & & \\ & \ddots & \ddots & \ddots & \\ & & b_{N-2}^{(\nu)} & a_{N-2}^{(\nu)} & c_{N-2}^{(\nu)} \\ & & & b_{N-1}^{(\nu)} & a_{N-1}^{(\nu)} \end{bmatrix}. \tag{6.3.30d}$$

Each Newton iteration requires the solution of a tridiagonal system. The Jacobian of this system need not be reevaluated and factored after each Newton step; thus, only the forward and backward substitution steps of the tridiagonal algorithm shown in Figure 6.3.2 need be performed at each iterative step. The derivatives $\partial f/\partial y$ and $\partial f/\partial y'$ can be approximated by finite differences.

Convergence of Newton's method is typically quadratic except at a bifurcation point where it is often linear. The use of finite difference approximations in the Jacobian also slows the convergence rate.

Example 6.3.3. Consider the elastica problem described in Section 6.1 and repeated here using the notation of this Section as

$$y'' + P \sin y = 0, \qquad y(0) = y(1/2) = 0.$$

Hence,

$$f(x, y, y') = -P \sin y,$$
$$\frac{\partial f}{\partial y} = -P \cos y, \qquad \frac{\partial f}{\partial y'} = 0,$$

and

$$b_i^{(\nu)} = c_i^{(\nu)} = 1, \qquad a_i^{(\nu)} = -2 + h^2 P \cos y_i^{(\nu)}.$$

Using the convergence criteria that

$$||\mathbf{F}(\mathbf{y}^{(\nu)})||_\infty = \max_{1 \le i \le N-1} |F_i(\mathbf{y}^{(\nu)})| \le 10^{-9}$$

and setting $P = 40$, we found the number of Newton iterations and solution at $x = 0.25$ to be as recorded in Table 6.3.2. The number of Newton iterations decreases as the mesh becomes finer. This is a result of the solution appearing to be smoother. The convergence rate seems to be nearly quadratic.

h	K	i	y_i
0.1	5	5	0.41240948
0.05	4	10	0.34795042
0.025	3	20	0.32985549

Table 6.3.2: Number of iterations K to reach convergence and the approximate solution at $x = 0.25$ for Example 6.3.3.

The development and description of finite difference equations may be simplified by introducing a set of finite difference operators as shown in Table 6.3.3.

The next several examples illustrate some applications of these finite difference operators.

Example 6.3.4. The centered difference formula (6.3.7) can be expressed in terms of the central difference and averaging operators δ and μ as

$$\frac{\mu\delta y_i}{h} = \frac{\mu(y_{i+1/2} - y_{i-1/2})}{h} = \frac{y_{i+1} - y_{i-1}}{2h}.$$

Example 6.3.5. An operator raised to a positive integer power is iterated, e.g.,

$$\delta^2 y_i = \delta(y_{i+1/2} - y_{i-1/2}) = y_{i+1} - 2y_i + y_{i-1}.$$

Thus, the centered second difference approximation (6.3.8) of the second derivative can be written as

$$y_i'' = \frac{\delta^2 y_i}{h^2}.$$

Example 6.3.6. Expanding $y(x_{i+1})$ in a Taylor's series about x_i yields

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) + \frac{h^3}{6}y'''(x_i) + \dots .$$

Using the derivative operator D defined in Table 6.3.3

$$y(x_{i+1}) = [1 + hD + \frac{h^2}{2}D^2 + \dots]y(x_i).$$

Operator	Symbol	Definition
Forward Difference	Δ	$\Delta y_i := y_{i+1} - y_i$
Backward Difference	∇	$\nabla y_i := y_i - y_{i-1}$
Central Difference	δ	$\delta y_i := y_{i+1/2} - y_{i-1/2}$
Average	μ	$\mu y_i := (y_{i+1/2} + y_{i-1/2})/2$
Shift	E	$E y_i := y_{i+1}$
Derivative	D	$D y_i := y_i'$

Table 6.3.3: Definition of finite difference operators.

This suggests the shorthand operator notation

$$Ey(x_i) = y(x_{i+1}) = e^{hD} y(x_i),$$

where E is the shift operator (Table 6.3.3). We, thus, infer the identity between the shift, exponential, and derivative operators

$$E = e^{hD}. \tag{6.3.31}$$

Additional relationships can be obtained by noting that $\Delta y_i = (E-1)y_i$, which implies that $\Delta = E - 1$ or $E = 1 + \Delta$. Using this with (6.3.31) gives

$$hD = \ln E = \ln(1+\Delta) = \Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \dots, \tag{6.3.32a}$$

where the series expansion of $\ln(1+x)$, $|x| < 1$, has been used. A similar relation in terms of the backward difference operator can be constructed by noting that $\nabla = 1 - E^{-1}$; thus,

$$hD = \ln E = -\ln(1-\nabla) = \nabla + \frac{1}{2}\nabla^2 + \frac{1}{3}\nabla^3 + \dots, \tag{6.3.32b}$$

These identities can be used to derive high-order finite difference approximations of first derivatives. For example, suppose that we retain the first two terms in (6.3.32a), i.e.,

$$hDy_i \approx [\Delta - \frac{1}{2}\Delta^2] y_i,$$

or

$$hDy_i \approx [(y_{i+1} - y_i) - \frac{y_{i+2} - 2y_{i+1} + y_i}{2}],$$

or

$$Dy_i \approx \frac{-y_{i+2} + 4y_{i+1} - 3y_i}{2h}.$$

This formula can be verified as an $O(h^2)$ approximation of $y'(x_i)$.

Example 6.3.7. Let us use (6.3.31) with h replaced by $\pm h/2$ to obtain

$$y(x_{i+1/2}) = e^{\frac{h}{2}D} y(x_i), \qquad y(x_{i-1/2}) = e^{-\frac{h}{2}D} y(x_i).$$

Subtracting the two formulas and using the central difference operator gives

$$\delta y(x_i) = (e^{\frac{h}{2}D} - e^{-\frac{h}{2}D}) y(x_i) = (2 \sinh \frac{h}{2} D) y(x_i).$$

Thus,

$$\delta = 2 \sinh \frac{h}{2} D,$$

or

$$hD = 2 \sinh^{-1} \frac{\delta}{2} = \delta - \frac{1}{2^2 3!}\delta^3 + \frac{3^2}{2^4 5!}\delta^5 - \dots, \tag{6.3.33}$$

which can be used to construct central difference approximations of $y'(x_i)$.

Example 6.3.8. We can square, cube, etc. relations (6.3.32) and (6.3.33) to construct approximations of second, third, *etc.* derivatives. For example, squaring (6.3.33) gives

$$h^2 D^2 y(x_i) = h^2 y''(x_i) = [\delta^2 - \frac{1}{12}\delta^4 + \frac{1}{90}\delta^6 + \dots] y(x_i). \tag{6.3.34}$$

At some point, these formal manipulations would have to be verified as being correct and estimates of their local discretization errors would have to be obtained. Nevertheless, using the formal operators of Table 6.3.3 provides us with a simple way of developing high-order finite difference approximations.

6.4 Introduction to Collocation Methods

Unlike finite difference methods, projection methods such as collocation give a continuous approximation of the solution as a function of x. The basic idea is to approximate the

solution $y(x)$ of a BVP by a simpler function $Y(x)$ and then determine $Y(x)$ so that it is the "best" approximation of $y(x)$ in some sense. Two reasonable choices for $Y(x)$ are a discrete Fourier series

$$Y(x) = \sum_{k=0}^{M-1} c_k e^{ikx}$$

and a polynomial

$$Y(x) = \sum_{k=0}^{M-1} c_k x^k.$$

It is convenient to regard the BVP solution $y(x)$ as an element of an infinite-dimensional function space and $Y(x)$ as an element of an M-dimensional subspace of it. Thus, assuming that $y(x)$ has continuous second derivatives on $a < x < b$, we would write $y(x) \in C^2(a,b)$ which is read "$y(x)$ is an element of the space of functions that have continuous second derivatives on (a,b)." Then, $Y(x) \in S^M \subset C^2(a,b)$ where the space S^M consists of those C^2 functions having a prescribed form. The chosen functions, e.g., e^{ikx} or x^k, $k = 0, 1, \ldots, M-1$, comprise a basis for S^M.

In order to introduce some concepts, we'll again focus on the second-order, nonlinear, scalar BVP (6.3.1). After selecting a basis, the "coordinates" c_k, $k = 0, 1, \ldots, M-1$, can be determined by, e.g., the least squares technique

$$\min_{Y \in s^M} \int_a^b R^2(x) dx$$

where $R(x)$ is the residual

$$R(x) = Y'' - f(x, Y, Y').$$

In this case, it is clear that $Y(x)$ is the "best" approximation of $y(x)$ in the sense of minimizing the square of the integral of the residual.

Using Galerkin's method, we determine $Y(x)$ so that the residual $R(x)$ is "orthogonal" to every function in S^M, i.e.,

$$\int_a^b w(x) R(x) dx = 0, \qquad \forall w(x) \in S^M.$$

The optimality of this procedure is not clear; however, since Galerkin's method is primariliy used with partial differential equations, we will not pursue it further.

Collocation has been shown to be a successful procedure tor two-point BVPs and is the one on which we focus. Collocation consists of satisfying

$$R(\xi_i) = 0, \qquad i = 1, 2, \ldots, M,$$

with

$$a \leq \xi_1 < \xi_2 < \ldots < \xi_M \leq b.$$

The optimality of collocation is also not clear, but we'll pursue this elsewhere.

Global approximations such as the Fourier series and the polynomials introduced above lead to ill-conditioned algebraic problems. It is far better to use piecewise polynomial approximations that result in sparse and well-conditioned algebraic systems. It is also unwise to infer more continuity than necessary. Discontinuous and continuous piecewise polynomial approximations might have the forms shown in Figure 6.4.1. The discontinuous polynomial (on the left) has jumps at x_i, $i = 1, 2, \ldots, N-1$. Thus, the first derivative doesn't exist at these points and this would be an unsuitable function to approximate the solution of a second-order ODE. The continuous approximation (on the right) has jumps in its first derivative at x_i, $i = 1, 2, \ldots, N-1$, and its second derivative doesn't exist at these points. Hence, minimally $Y(x) \in C^1(a, b)$. In this case, the first derivative of $Y(x)$ is continuous and the second derivative is piecewise continuous.

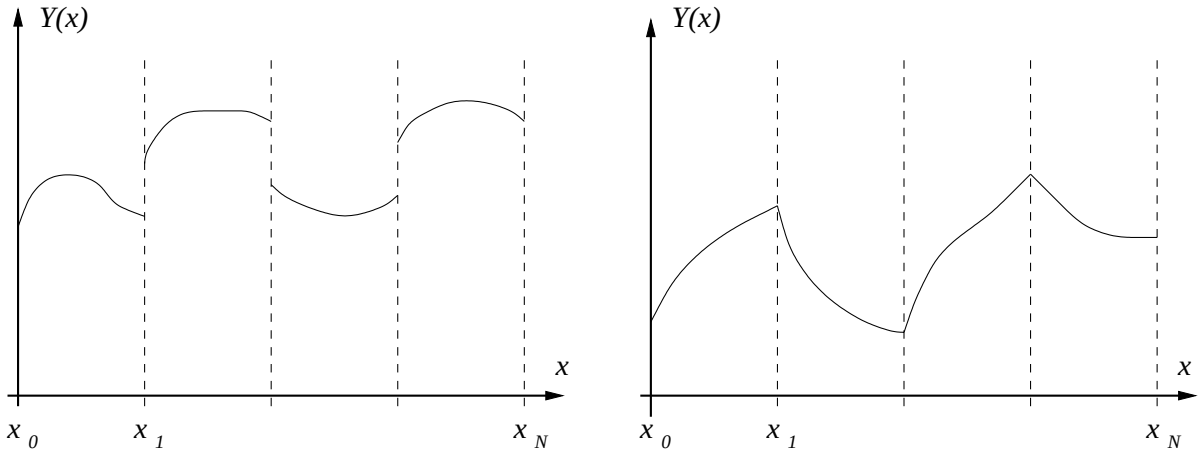

Figure 6.4.1: Discontinuous (left) and continuous (right) piecewise polynomial function $Y(x)$ having jumps (left) and jumps in its first derivative (right) at x_i, $i = 1, 2, \ldots, N-1$.

Perhaps the simplest way of satisfying the continuity requirements is to select a basis for S^M that includes them. For example, a basis for a space of piecewise constant

functions could be chosen as

$$\phi_i^0(x) = \begin{cases} 1, & \text{if } x \in [x_{i-1}, x_i) \\ 0, & \text{otherwise} \end{cases}, \qquad i = 1, 2, \ldots, N. \tag{6.4.1}$$

The approximation $Y(x)$ would then be written in the form

$$Y(x) = \sum_{i=1}^{N} c_i \phi_i^0(x) \tag{6.4.2}$$

and is shown in Figure 6.4.2. (In this case, the dimension of the subspace M and the number of subintervals N are identical; however, this need not be so.)

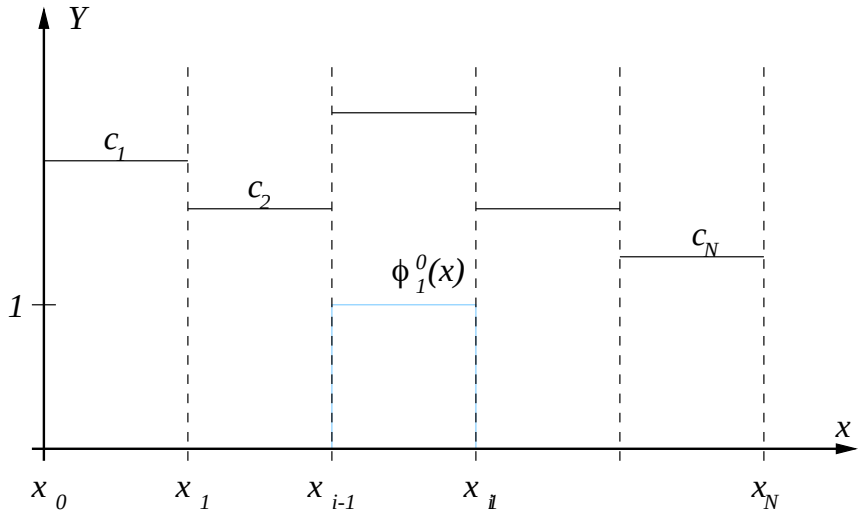

Figure 6.4.2: Piecewise constant basis element $\phi_i^0(x)$ and the resulting piecewise constant approximation $Y(x)$.

Using (6.4.1) and (6.4.2) we see that

$$Y(x) = c_i, \qquad x \in [x_{i-1}, x_i).$$

We may interpret c_i as $Y(x_{i-1/2})$; however, this is not necessary. As shown in Figure 6.4.2, the basis $\phi_i^0(x) \in C^{-1}[a, b)$; thus, $Y(x) \in C^{-1}[a, b)$.

Of course, the basis (6.4.1) doesn't satisfy any continuity requirements; however, it can be used to generate a continuous basis. More generally, a piecewise-polynomial basis whose continuity increases with increasing degree can be constructed by integrating a linear combination of basis elements of a piecewise-polynomial space having one degree

less than the desired degree. For example, let us construct a C^0 piecewise-linear basis by integrating ϕ_i^0 and ϕ_{i+1}^0 as

$$\phi_i^1(x) = \int_{x_0}^{x} [\alpha\phi_i^0(s) + \beta\phi_{i+1}^0(s)]ds.$$

The appropriate continuity will be automatically obtained by the integration. The result for this example is

$$\phi_i^1(x) = \begin{cases} 0, & \text{if } x < x_{i-1} \\ \alpha(x - x_{i-1}), & \text{if } x_{i-1} \le x < x_i \\ \alpha h_i + \beta(x - x_i), & \text{if } x_i \le x < x_{i+1} \\ \alpha h_i + \beta h_{i+1}, & \text{if } x_{i+1} \le x \end{cases}$$

where

$$h_i = x_i - x_{i-1}. \tag{6.4.3}$$

The parameters α and β are at our disposal. Let us pick them so that the resulting approximation has compact support, i.e., so that

$$\phi_i^1(x) = 0, \qquad x \ge x_{i+1}.$$

Let us, furthermore normalize the basis so that

$$\phi_i^1(x_i) = 1.$$

Then, we find

$$\alpha_i = \frac{1}{h_i}, \qquad \beta_i = -\frac{1}{h_{i+1}}$$

and the final result

$$\phi_i^1(x) = \begin{cases} \frac{x - x_{i-1}}{h_i}, & \text{if } x_{i-1} \le x < x_i \\ \frac{x_{i+1} - x}{h_{i+1}}, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases} . \tag{6.4.4}$$

The approximation $Y(x)$ has the form

$$Y(x) = \sum_{i=0}^{N} c_i \phi_i^1(x). \tag{6.4.5}$$

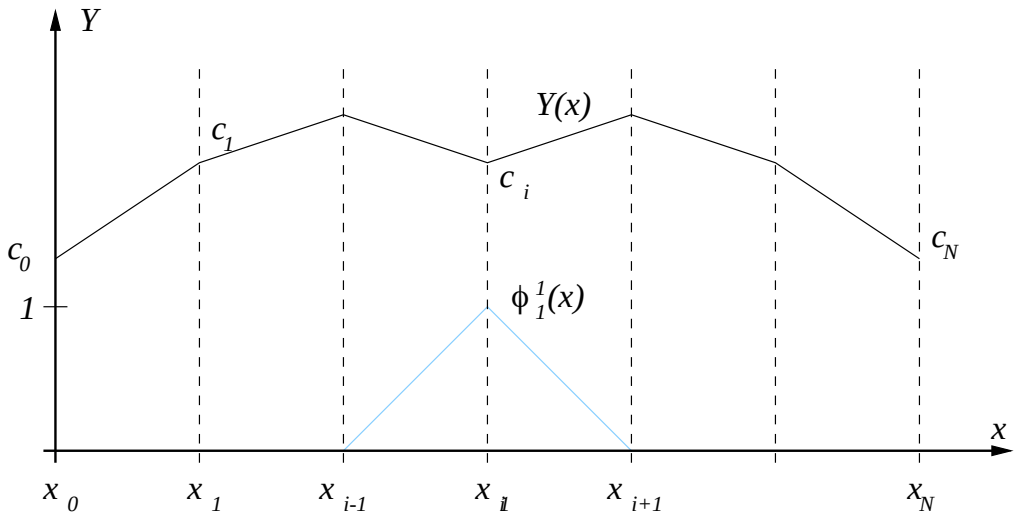

Figure 6.4.3: Piecewise-linear basis element $\phi_i^1(x)$ and the resulting piecewise-linear approximation.

The basis element $\phi_i^1(x)$ and the piecewise-linear approximation $Y(x)$ are shown in Figure 6.4.3. Using (6.4.4) we see that $\phi_i^1(x_j) = \delta_{ij}$ where δ_{ij} is the Kronecker delta. Using this with (6.4.5) yields $Y(x_i) = c_i$, $i = 0, 1, \ldots, N$. The restriction of $Y(x)$ to the subinterval $[x_{i-1}, x_i)$ is the linear function

$$Y(x) = c_{i-1}\phi_{i-1}^1(x) + c_i\phi_i^1(x), \qquad x \in [x_{i-1}, x_i).$$

We'll continue by constructing a C^1 piecewise-quadratic polynomial basis as

$$\phi_i^2(x) = \int_{x_0}^{x} [\alpha\phi_{i-1}^1(s) + \beta\phi_i^1(s)]ds.$$

Proceeding in three steps, we see that

$$\int_{x_0}^{x} \phi_i^1(s)ds = \begin{cases} 0, & \text{if } x < x_{i-1} \\ \frac{(x-x_{i-1})^2}{2h_i}, & \text{if } x_{i-1} \le x < x_i \\ \frac{h_i}{2} + \frac{h_{i+1}}{2} - \frac{(x_{i+1}-x)^2}{2h_{i+1}}, & \text{if } x_i \le x < x_{i+1} \\ \frac{h_i}{2} + \frac{h_{i+1}}{2}, & \text{if } x_{i+1} \le x \end{cases}$$

and

$$\phi_i^2(x) = \frac{1}{2}\begin{cases} 0, & \text{if } x_{i-2} < x \\ \alpha\frac{(x-x_{i-2})^2}{h_{i-1}}, & \text{if } x_{i-2} \le x < x_{i-1} \\ \alpha[h_{i-1} + h_i - \frac{(x_i-x)^2}{h_i}] + \beta\frac{(x-x_{i-1})^2}{h_i}, & \text{if } x_{i-1} \le x < x_i \\ \alpha(h_{i-1} + h_i) + \beta[h_i + h_{i+1} - \frac{(x_{i+1}-x)^2}{h_{i+1}}], & \text{if } x_i \le x < x_{i+1} \\ \alpha(h_{i-1} + h_i) + \beta(h_i + h_{i+1}), & \text{if } x_{i+1} \le x \end{cases}.$$

Enforcing the condition that $\phi_i^2(x) = 0$, $x \geq x_{i+1}$ implies

$$\alpha = \frac{2\gamma}{h_{i-1} + h_i}, \qquad \beta = -\frac{2\gamma}{h_i + h_{i+1}}.$$

Thus,

$$\phi_i^2(x) = \gamma \begin{cases} \frac{(x-x_{i-2})^2}{h_{i-1}(h_{i-1}+h_i)}, & \text{if } x_{i-2} \leq x < x_{i-1} \\ 1 - \frac{(x_i-x)^2}{h_i(h_{i-1}+h_i)} - \frac{(x-x_{i-1})^2}{h_i(h_i+h_{i+1})}, & \text{if } x_{i-1} \leq x < x_i \\ \frac{(x_{i+1}-x)^2}{h_{i+1}(h_i+h_{i+1})}, & \text{if } x_i \leq x < x_{i+1} \\ 0, & \text{otherwise} \end{cases} . \tag{6.4.6a}$$

Normalizing the result so that $\phi_i^2(x_{i-1/2}) = 1$ yields

$$\gamma = \left[1 - \frac{h_i}{4(h_{i-1} + h_i)} - \frac{h_i}{4(h_i + h_{i+1})}\right]^{-1}. \tag{6.4.6b}$$

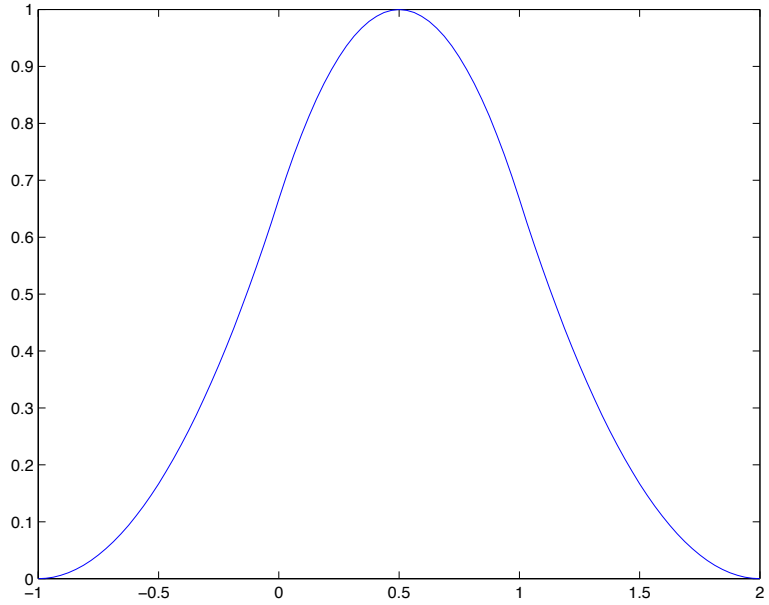

Figure 6.4.4: Quadratic spline basis element $\phi_i^2(x)$ relative to a mesh with $x_{i-2} = -1$, $x_{i-1} = 0$, $x_i = 1$, and $x_{i+1} = 2$.

The basis $\{\phi_i^2(x)\}_{i=1}^N$ defines a quadratic spline. The element ϕ_i^2 is shown in Figure 6.4.4. Evaluating (6.4.6a,b) at a node x_j yields

$$\phi_i^2(x_j) = \gamma \begin{cases} \frac{h_{i-1}}{h_{i-1}+h_i}, & \text{if } j = i-1 \\ \frac{h_{i+1}}{h_i+h_{i+1}}, & \text{if } j = i \\ 0, & \text{otherwise} \end{cases} . \tag{6.4.6c}$$

The basis simplifies when the mesh is uniform; thus, if $h_i = h$, $i = 1, 2, \ldots, N$,

$$\phi_i^2(x) = \frac{2}{3} \begin{cases} (\frac{x - x_{i-2}}{h})^2, & \text{if } x_{i-2} \le x < x_{i-1} \\ 2 - (\frac{x_i - x}{h})^2 - (\frac{x - x_{i-1}}{h})^2, & \text{if } x_{i-1} \le x < x_i \\ (\frac{x_{i+1} - x}{h})^2, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases} . \tag{6.4.6d}$$

The quadratic spline approximation is written in the form

$$Y(x) = \sum_{i=1}^{N} c_i \phi_i^2(x) \tag{6.4.7a}$$

and its restriction to $[x_{i-1}, x_i)$ is

$$Y(x) = \begin{cases} c_1\phi_1^2(x) + c_2\phi_2^2(x), & x \in [x_0, x_1) \\ c_{i-1}\phi_{i-1}^2(x) + c_i\phi_i^2(x) + c_{i+1}\phi_{i+1}^2(x), & x \in [x_{i-1}, x_i),\ i \ne 1, N \\ c_{N-1}\phi_{N-1}^2(x) + c_N\phi_N^2(x), & x \in [x_{N-1}, x_N) \end{cases} . \tag{6.4.7b}$$

Thus, three elements of the spline basis are nonzero on any interval except the first and the last where there are only two nonzero elements.

Using (6.4.6) with (6.4.7b) we see that

$$Y(x_i) = \gamma \begin{cases} \frac{c_1 h_0}{h_0 + h_1}, & \text{if } i = 1 \\ \frac{c_i h_{i+1} + c_{i+1} h_i}{h_i + h_{i+1}}, & \text{if } i = 2, 3, \ldots, N-1 \\ \frac{c_N h_N}{h_N + h_{N+1}}, & \text{if } i = N \end{cases} . \tag{6.4.7c}$$

One could take $h_0 = h_1$ and $h_{N+1} = h_N$.

When solving BVPs or interpolation and approximation problems, it's convenient to introduce an extra basis element and unknown at each end of the domain and write $Y(x)$ as

$$Y(x) = \sum_{i=0}^{N+1} c_i \phi_i^2(x). \tag{6.4.8}$$

Now there are three nonzero basis elements on every subinterval. For interpolation problems, the coefficients c_i, $i = 0, 1, \ldots, N+1$, may be determined by satisfying

$$Y(x_{i-1}/2) = f(x_{i-1}/2), \qquad i = 1, 2, \ldots, N.$$

This yields N equations for the $N + 2$ unknowns. The extra two equations could be obtained by

- interpolating $f(x)$ at x_0 and x_N,

- prescribing $f'(x)$ at x_0 and x_N, or
- prescribing $Y'(x_0) = Y'(x_N) = 0$.

Regardless of the boundary prescription, this interpolation problem requires the solution of a tridiagonal linear algebraic problem to determine c_i, $i = 0, 1, \ldots, N+1$.

It's possible to continue in this manner, introducing more continuity with increasing polynomial degree; however, usually we'll want approximations having the minimum allowable continuity. Since higher-degree approximations increase the convergence rate of smooth solutions, we seek alternative spline constructions that do not increase smoothness with increasing polynomial degree. Such procedures exist [3]; however, for the moment, we'll examine piecewise cubic Hermite approximation. Thus, let us consider a function of the form

$$Y(x) = \sum_{i=0}^{N} [c_i \phi_i^3(x) + d_i \omega_i^3(x)]. \tag{6.4.9}$$

The basis $\{\phi_i^3(x), \omega_i^3(x)\}_{i=0}^N$ is constructed to satisfy the following conditions:

1. $\phi_i^3(x)$ and $\omega_i^3(x)$ are piecewise cubic polynomials on (x_0, x_N),
2. $\phi_i^3(x), \omega_i^3(x) \in C^1(x_0, x_N)$,
3. $\phi_i^3(x)$ and $\omega_i^3(x)$ are nonzero only on $[x_{i-1}, x_{i+1})$, and
4. $\phi_i^3(x_i) = 1$ and $\omega_i^3(x_i) = 0$, $i = 0, 1, \ldots, N$.

These conditions imply that

$$\phi_i^3(x_j) = \delta_{ij}, \qquad \phi_i'^3(x_j) = 0, \qquad i, j = 0, 1, \ldots, N, \tag{6.4.10a}$$

$$\omega_i^3(x_j) = 0, \qquad \omega_i'^3(x_j) = \delta_{ij}, \qquad i, j = 0, 1, \ldots, N. \tag{6.4.10b}$$

Together, (6.4.9) and (6.4.10) give $Y(x_i) = c_i$ and $Y'(x_i) = d_i$, $i = 0, 1, \ldots, N$.

Given these requirements, the piecewise cubic Hermite basis is

$$\phi_i^3(x) = \begin{cases} 1 - 3(\frac{x-x_i}{h_i})^2 - 2(\frac{x-x_i}{h_i})^3, & \text{if } x_{i-1} \le x < x_i \\ 1 - 3(\frac{x-x_i}{h_{i+1}})^2 + 2(\frac{x-x_i}{h_{i+1}})^3, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases}, \tag{6.4.11a}$$

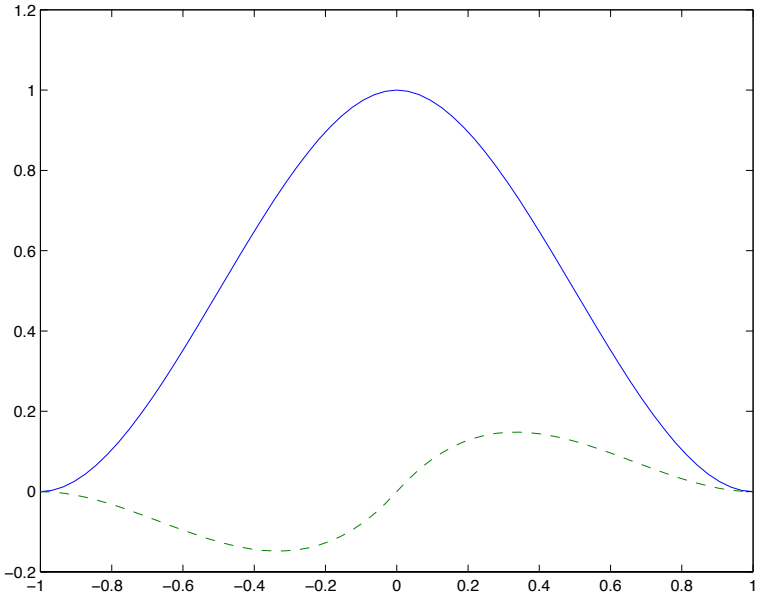

Figure 6.4.5: Cubic Hermite polynomial basis elements $\phi_i^3(x)$ (solid) and $\omega_i^3(x)$ (dashed) relative to a mesh with $x_{i-1} = -1$, $x_i = 0$, and $x_{i+1} = 1$.

$$\omega_i^3(x) = \begin{cases} (x - x_i)[1 + \frac{x-x_i}{h_i}]^2, & \text{if } x_{i-1} \le x < x_i \\ (x - x_i)[1 - \frac{x-x_i}{h_{i+1}}]^2, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases} . \tag{6.4.11b}$$

Representative basis elements are illustrated in Figure 6.4.5. Examining (6.4.9), we see that there are four nontrivial basis elements ϕ_{i-1}, ω_{i-1}, ϕ_i, and ω_i on the subinterval (x_{i-1}, x_i), $i = 1, 2, \ldots, N$.

Having constructed appropriate piecewise polynomial approximations, let us use them to define a collocation method by satisfying (6.3.1) at a prescribed number of points per subinterval and (typically) satisfying the boundary conditions. Thus,

$$Y''(\xi_{ij}) = f(\xi_{ij}, Y(\xi_{ij}), Y'(\xi_{ij})), \qquad j = 1, 2, \ldots, J, \qquad i = 1, 2, \ldots, N, \tag{6.4.12a}$$

$$Y(a) = A, \qquad Y(b) = B. \tag{6.4.12b}$$

As indicated, there are J collocation points ξ_{ij}, $j = 1, 2, \ldots, J$, per subinterval. For the piecewise quadratic spline approximation (6.4.8), we would determine the $N + 2$ unknowns c_i, $i = 0, 1, \ldots, N$, by collocating at $J = 1$ point per subinterval and satisfying the boundary conditions (6.4.12b). With the piecewise cubic Hermite polynomial (6.4.9),

we would determine the $2(N+1)$ unknowns c_i, d_i, $i = 0, 1, \ldots, N$, by collocating at $J = 2$ point per subinterval and satisfying (6.4.12b).

Example 6.4.1. We'll develop the collocation equations when piecewise quadratic splines (6.4.8) are applied to a linear problem with $f(x, y, y')$ given by (6.3.13). For simplicity, we'll assume that the mesh is uniform with spacing h. Utilizing (6.4.6) and (6.4.8), the boundary conditions (6.4.12b) are

$$Y(a) = A = \frac{2}{3}(c_0 + c_1), \qquad Y(b) = B = \frac{2}{3}(c_N + c_{N+1}). \tag{6.4.13a}$$

Selecting the sole collocation point

$$\xi_{i,1} = x_{i-1/2} = \frac{x_{i-1} + x_i}{2}, \qquad i = 1, 2, \ldots, N,$$

at the center of each subinterval (Figure 6.4.6) and using (6.4.6) and (6.4.8) we have

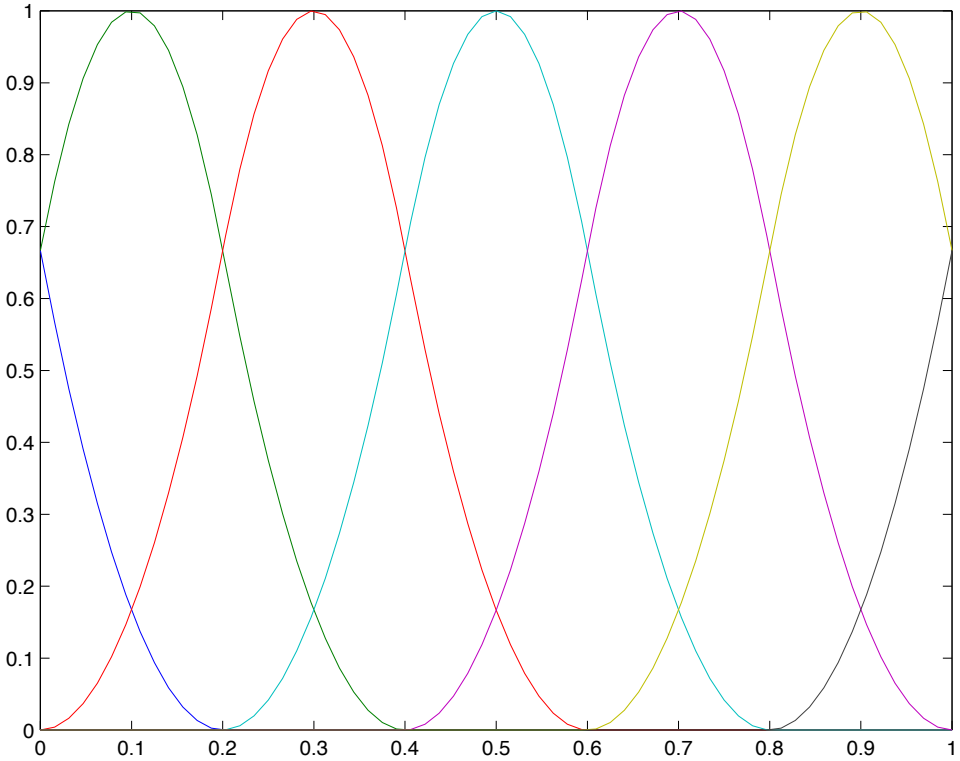

Figure 6.4.6: Piecewise quadratic spline basis used for collocation on a mesh with spacing $h = 0.2$ on $[0, 1]$.

$$Y(x_{i-1/2}) = \frac{1}{6}(c_{i-1} + 6c_i + c_{i+1}),$$

$$Y'(x_{i-1/2}) = \frac{2}{3h}(c_{i+1} - c_{i-1}),$$

and

$$Y''(x_{i-1/2}) = \frac{4}{3h^2}(c_{i-1} - 2c_i + c_{i+1}).$$

Substituting these results into (6.4.12a) while using (6.3.13), we find

$$\frac{4}{3h^2}(c_{i-1} - 2c_i + c_{i+1}) + \frac{2p_{i-1/2}}{3h}(c_{i+1} - c_{i-1}) + \frac{q_{i-1/2}}{6}(c_{i-1} + 6c_i + c_{i+1}) = r_{i-1/2},$$
$$i = 1, 2, \ldots, N. \qquad (6.4.13b)$$

These difference equations are similar, but not identical, to the central finite difference equations (6.3.14). The relationship can be made more precise by re-writing (6.4.13) in terms of the nodal unknowns $Y(x_i)$ and comparing this result with (6.3.14) (Problem 2).

The system (6.4.13) may be written in a tridiagonal form as

$$\begin{bmatrix} \alpha_0 & \gamma_0 & & & \\ \beta_1 & \alpha_1 & \gamma_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_N & \alpha_N & \gamma_N \\ & & & \beta_{N+1} & \alpha_{N+1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \\ c_{N+1} \end{bmatrix} = \begin{bmatrix} A \\ r_1 \\ \vdots \\ r_N \\ B \end{bmatrix}, \qquad (6.4.14a)$$

where

$$\alpha_0 = \alpha_{N+1} = \frac{2}{3}, \qquad \alpha_i = -\frac{8}{3h^2} + q_{i-1/2}, \qquad i = 1, 2, \ldots, N, \qquad (6.4.14b)$$

$$\beta_i = \frac{4}{3h^2} - \frac{2p_{i-1/2}}{3h} + \frac{q_{i-1/2}}{6}, \qquad i = 1, 2, \ldots, N, \qquad \beta_{N+1} = \frac{2}{3}, \qquad (6.4.14c)$$

$$\gamma_0 = \frac{2}{3}, \qquad \gamma_i = \frac{4}{3h^2} + \frac{2p_{i-1/2}}{3h} + \frac{q_{i-1/2}}{6}, \qquad i = 1, 2, \ldots, N. \qquad (6.4.14d)$$

As a simple numerical example, let's suppose $p = 0$, $q = -1$, $r = 0$, $A = 0$, and $B = 1$. This is the problem of Example 6.3.2 which has the exact solution

$$y(x) = \frac{\sinh x}{\sinh 1}.$$

Solution and pointwise (at x_i, $i = 0, 1, \ldots, N$) error data are presented in Table 6.4.1 for $N = 10$. The maximum pointwise errors for collocation solutions computed with

i	x_i	$Y(x_i)$	$y(x_i) - Y(x_i)$
0	0.0	0.0000000	0.0000000E+00
1	0.1	0.0852282	0.5491078E-05
2	0.2	0.1713098	0.1069903E-04
3	0.3	0.2591066	0.1528859E-04
4	0.4	0.3494977	0.1892447E-04
5	0.5	0.4433881	0.2130866E-04
6	0.6	0.5417180	0.2211332E-04
7	0.7	0.6454719	0.2080202E-04
8	0.8	0.7556885	0.1698732E-04
9	0.9	0.8734714	0.1043081E-04
10	1.0	1.0000000	0.2384186E-06

Table 6.4.1: Solution and pointwise errors for Example 6.4.1 using collocation with piecewise quadratic splines at the center of each of ten subintervals.

N	$\|\|Y - y\|\|_\infty$	$N^2\|\|Y - y\|\|_\infty$
5	0.8890263×10^{-4}	0.002223
10	0.2214184×10^{-4}	0.002214
20	0.5537689×10^{-5}	0.002215

Table 6.4.2: Maximum pointwise errors for the solution of Example 6.4.1 using collocation with piecewise quadratic splines at the center of each of N subintervals.

$N = 5, 10$, and 20 subintervals are presented in Table 6.4.2. Solutions appear to be converging as $O(h^2)$. Pointwise errors are about half of those found using central finite differences (Table 6.3.1).

Example 6.4.2. In the previous example, it seemed natural to place the single collocation point at the center of each subinterval. Were we to used piecewise cubic Hermite approximations, however, we would have two collocation points per subinterval. Placing them at the ends of each subinterval is one possibility; however, our work with implicit Runge-Kutta methods (Section 3.3) would suggest that the Gauss-Legendre points give a higher rate of convergence. DeBoor and Swartz [2] showed that this is the case and we will repeat this analysis in Chapter 9; however, for the moment, let us assume it so and consider the collocation solution of ([1], Chapter 5)

$$y'' + \frac{y'}{x} = (\frac{8}{8 - x^2})^2, \qquad 0 < x < 1,$$

$$y'(0) = y(1) = 0,$$

N	$\|\|Y-y\|\|_\infty$
2	0.20×10^{-3}
5	0.64×10^{-5}
10	0.46×10^{-6}
20	0.33×10^{-7}
40	0.23×10^{-8}
80	0.16×10^{-9}

Table 6.4.3: Maximum pointwise errors for the solution of Example 6.4.2 using collocation with piecewise cubic Hermite polynomials at two Gauss-Legendre points per subinterval.

which has the exact solution

$$y(x) = 2\ln(\frac{7}{8-x^2}).$$

The solution is smooth on $0 \le x \le 1$, but the coefficient $p(x) = 1/x$ is unbounded at $x = 0$. This would lead to problems with finite difference or shooting methods, but not with collocation methods that collocate at points other than subinterval ends. Ascher *et al.* [1] solve this problem by a variety of techniques. We'll report their results using piecewise cubic Hermite polynomials with collocation at the two Gauss-Legendre points

$$\xi_{i,1} = \frac{1}{2}[(x_{i-1}+x_i) - \frac{h}{\sqrt{3}}], \qquad \xi_{i,2} = \frac{1}{2}[(x_{i-1}+x_i) + \frac{h}{\sqrt{3}}]$$

per subinterval. The maximum pointwise errors are presented in Table 6.4.3. A simple calculation verifies that the solution is converging as $O(N^{-4})$.

Problems

1. Construct the basis for a cubic spline approximation that is of class C^2 and has support on $[x_{i-2}, x_{i+2}]$ by integrating the quadratic splines $\phi_i^2(x)$ and $\phi_{i+1}^2(x)$ of (6.4.6) and imposing appropriate normalization and compact support conditions. For simplicity, assume that the mesh is uniform with spacing h.

2. Using (6.4.7c) show that

$$Y(x_i) = \frac{2}{3}(c_i + c_{i+1})$$

on a uniform mesh of spacing h. Use this to rewrite the quadratic-spline collocation equations (6.4.13) in terms of $Y(x_i)$, $i = 1, 2, \ldots, N$, instead of c_i, $i = 0, 1, \ldots, N+1$. Compare the results with the finite difference equations (6.3.14).

Bibliography

[1] U.M. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM, Philadelphia, second edition, 1995.

[2] C. de Boor and B. Swartz. Collocation at gaussian points. *SIAM J. Numer. Anal.*, 10:582–687, 1973.

[3] C. deBoor. *A Practical Guide to Splices*. Springer-Verlag, Berlin, 1978.

Chapter 7

Initial Value Methods

7.1 Introduction

We'd like to extend the initial-value or shooting procedures to vector systems of m first-order ODEs. However, before studying nonlinear problems, let's focus on the linear system

$$\mathbf{y}' = \mathbf{A}\mathbf{y} + \mathbf{b} \tag{7.1.1a}$$

with separated boundary conditions.

$$\mathbf{L}\mathbf{y}(a) = \mathbf{l}, \tag{7.1.1b}$$

$$\mathbf{R}\mathbf{y}(b) = \mathbf{r} \tag{7.1.1c}$$

We'll suppose that $\mathbf{A}$ is $m \times m$, $\mathbf{L}$ is $l \times m$, $\mathbf{R}$ is $r \times m$, $\mathbf{b}$ is an m-vector, $\mathbf{l}$ is an l-vector, and $\mathbf{r}$ is an r-vector. Of course, $l + r = m$.

It's simple to develop shooting procedures be realizing that the solution of the linear BVP (7.1.1) can be written in the form [3, 4]

$$\mathbf{y}(x) = \mathbf{Y}(x)\mathbf{c} + \mathbf{v}(x). \tag{7.1.2a}$$

The function $\mathbf{v}(x)$ is a *particular solution* satisfying, e.g.,

$$\mathbf{v}' = \mathbf{A}\mathbf{v} + \mathbf{b}, \qquad \mathbf{v}(a) = \boldsymbol{\alpha}, \tag{7.1.2b}$$

for some convenient choice of the vector $\boldsymbol{\alpha}$ (e.g., $\boldsymbol{\alpha} = \mathbf{0}$). The $m \times m$ matrix $\mathbf{Y}(x)$ is a set of *fundamental solutions* satisfying

$$\mathbf{Y}' = \mathbf{A}\mathbf{Y}, \qquad \mathbf{Y}(a) = \mathbf{I}, \tag{7.1.2c}$$

where $\mathbf{I}$ is the $m \times m$ identity matrix. The constant vector $\mathbf{c}$ is determined by satisfying the initial and terminal conditions (7.1.1b,c) give

$$\begin{bmatrix} \mathbf{L}\mathbf{y}(a) \\ \mathbf{R}\mathbf{y}(b) \end{bmatrix} = \mathbf{Q}\mathbf{c} + \begin{bmatrix} \mathbf{L}\mathbf{v}(a) \\ \mathbf{R}\mathbf{v}(b) \end{bmatrix} = \begin{bmatrix} \mathbf{l} \\ \mathbf{r} \end{bmatrix}. \tag{7.1.3a}$$

The solution exists since

$$\mathbf{Q} = \begin{bmatrix} \mathbf{L}\mathbf{Y}(a) \\ \mathbf{R}\mathbf{Y}(b) \end{bmatrix} = \begin{bmatrix} \mathbf{L} \\ \mathbf{R}\mathbf{Y}(b) \end{bmatrix} \tag{7.1.3b}$$

has full rank m for a well-posed problem. Once $\mathbf{c}$ has been determined, the solution of the BVP is determined by (7.1.2a). The procedure requires the solution of the $m + 1$ vector IVPs (7.1.2b,c).

Although this technique seems straight forward, we recall that the stability of BVPs is quite different from that for IVPs. An IVP for (7.1.1a) is stable when the eigenvalues of $\mathbf{A}$ have non-positive real parts. This is generally not the case for BVPs where the eigenvalues of $\mathbf{A}$ for stable problems can have positive and negative real parts. Let's explore a simple example.

Example 7.1.1 ([1], Section 4.2). Consider the second-order linear BVP

$$y'' - R^2 y = 1, \qquad y(0) = y(1) = 0,$$

and write this as a first-order system by letting

$$y_1 = y, \qquad y_2 = \frac{y'}{R}.$$

This leads to the symmetric system

$$\mathbf{y}' = \begin{bmatrix} 0 & R \\ R & 0 \end{bmatrix} \mathbf{y} + \begin{bmatrix} 0 \\ 1/R \end{bmatrix}$$

with the initial and terminal conditions

$$[1\ 0]\mathbf{y}(0) = 0, \qquad [1\ 0]\mathbf{y}(1) = 0.$$

We'll let the particular solution satisfy

$$\mathbf{v}' = \begin{bmatrix} 0 & R \\ R & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 1/R \end{bmatrix}, \qquad \mathbf{v}(0) = \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}.$$

Thus,

$$\mathbf{v}(x) = \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}.$$

The set of fundamental solutions satisfies

$$\mathbf{Y}' = \begin{bmatrix} 0 & R \\ R & 0 \end{bmatrix} \mathbf{Y}, \qquad \mathbf{Y}(0) = \mathbf{I}.$$

The matrix

$$\mathbf{A} = \begin{bmatrix} 0 & R \\ R & 0 \end{bmatrix}$$

has the eigenvalues $\lambda = \pm R$; thus, an IVP would be unstable. We'll have to investigate the stability of the BVP.

We write the fundamental solution as

$$\mathbf{Y}(x) = \begin{bmatrix} \cosh Rx & \sinh Rx \\ \sinh Rx & \cosh Rx \end{bmatrix}.$$

From (7.1.2a), the solution of this BVP has the form

$$\mathbf{y}(x) = \begin{bmatrix} \cosh Rx & \sinh Rx \\ \sinh Rx & \cosh Rx \end{bmatrix} \mathbf{c} + \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}.$$

The initial and terminal conditions (7.1.3) are

$$\begin{bmatrix} 1 & 0 \\ \cosh R & \sinh R \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} -1/R^2 \\ -1/R^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Thus,

$$\mathbf{c} = \frac{1}{R^2} \begin{bmatrix} 1 \\ \frac{1-\cosh R}{\sinh R} \end{bmatrix},$$

and the solution of the BVP is

$$\mathbf{y}(x) = \frac{1}{R^2} \begin{bmatrix} \cosh Rx & \sinh Rx \\ \sinh Rx & \cosh Rx \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1-\cosh R}{\sinh R} \end{bmatrix} + \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}.$$

Difficulties arise when $R \gg 1$. In this case, the solution $y_1(x)$ is asymptotically given by

$$y_1 \approx \frac{1}{R^2}[e^{-Rx} + e^{-R(1-x)} - 1].$$

Thus, it is approximately $-1/R^2$ away from the boundaries at $x = 0$ and 1 with narrow boundary layers near both ends.

When Rx is large, we would not be able to distinguish between $\sinh Rx$ and $\cosh Rx$ and, instead of the exact result, we would compute

$$\mathbf{y}(x) \approx \frac{1}{2} \begin{bmatrix} e^{Rx} & e^{Rx} \\ e^{Rx} & e^{Rx} \end{bmatrix} \mathbf{c} + \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}, \qquad x > 0.$$

The matrix $\mathbf{Y}(x)$ would be singular in this case. With small discernable differences between $\sinh Rx$ and $\cosh Rx$, $\mathbf{Y}$ would be ill-conditioned. Assuming this to be so, let's write the solution as

$$\mathbf{y}(x) \approx \frac{1}{2} \begin{bmatrix} e^{Rx}(1+\delta) & e^{Rx}(1-\delta) \\ e^{Rx}(1-\delta) & e^{Rx}(1+\delta) \end{bmatrix} \mathbf{c} + \begin{bmatrix} -1/R^2 \\ 0 \end{bmatrix}, \qquad x > 0,$$

where $\delta \ll 1$. (If $\delta = e^{-2R}$ then the above relation is exact at $x = 1$.) We would then determine $\mathbf{c}$ as

$$\mathbf{c} \approx \frac{1}{R^2} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

and the solution as

$$\mathbf{y}(x) \approx \frac{\delta e^{Rx}}{2R^2} \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad x > 0.$$

With R large, there is the possibility of having catastrophic growth in the solution even when δ is small. In order to demonstrate this, we solved this problem using the MATLAB Runge-Kutta procedure ode45. While an explicit Runge-Kutta code is not the most efficient solution procedure for this problem, efficiency was not our main concern. The maximum errors in the solution component y_1 with $R = 1, 10$ are reported in Table 7.1.1. The error has grown by six decades for a ten-fold increase in R. Indeed, the procedure failed to find a solution of the BVP with $R = 100$.

R	$\|e_1\|_\infty / \|y_1\|_\infty$
1	2.445×10^{-8}
10	2.940×10^{-2}

Table 7.1.1: Maximum errors in y_1 for Example 7.1.1

Let's pursue this difficulty with a bit more formality. The exact solution of the linear

BVP (7.1.1) can be written as (Problem 1)

$$\mathbf{y}(x) = \mathbf{Y}(x)\mathbf{Q}^{-1}\begin{bmatrix}\mathbf{l}\\ \mathbf{r}\end{bmatrix} + \int_a^b \mathbf{G}(x,\xi)\mathbf{b}(\xi)d\xi \tag{7.1.4a}$$

where $\mathbf{Q}$ was given by (7.1.3b) and $\mathbf{G}(x,\xi)$ is the *Green's function*

$$\mathbf{G}(x,\xi) = \begin{cases} \mathbf{Y}(x)\mathbf{Q}^{-1}\begin{bmatrix}\mathbf{L}\\ \mathbf{0}\end{bmatrix}\mathbf{Y}(a)\mathbf{Y}^{-1}(\xi), & \text{if } \xi \le x \\ -\mathbf{Y}(x)\mathbf{Q}^{-1}\begin{bmatrix}\mathbf{0}\\ \mathbf{R}\end{bmatrix}\mathbf{Y}(b)\mathbf{Y}^{-1}(\xi), & \text{if } \xi > x \end{cases}. \tag{7.1.4b}$$

The Green's function behaves like the inverse of the differential operator. We may use it to define the stability of BVP.

Definition 7.1.1. *The BVP (7.1.1) is stable if there exists a constant κ such that*

$$\|\mathbf{y}(\cdot)\|_\infty \le \kappa[|\mathbf{l}|_\infty + |\mathbf{r}|_\infty + \int_a^b |\mathbf{b}(\xi)|_\infty d\xi] \tag{7.1.5a}$$

where

$$\|\mathbf{y}(\cdot)\|_\infty = \max_{a\le x\le b}|\mathbf{y}(x)|_\infty, \qquad |\mathbf{b}|_\infty = \max_{1\le i\le m}|b_i|. \tag{7.1.5b}$$

From (7.1.4a), we see that

$$\kappa = \max(\|\mathbf{Y}(\cdot)\mathbf{Q}^{-1}\|_\infty, \|\mathbf{G}(\cdot,\cdot)\|_\infty) \tag{7.1.5c}$$

assuming that all quantities are bounded. Thus, the solution of a stable BVP is bounded by its data. Following the techniques used for IVPs, we may show that this also applies to a perturbation of the data associated with the BVP (7.1.1) ([2], Chapter 6); hence, the reason for the name stability.

Definition 7.1.2. *The BVP (7.1.1) has exponential dichotomy if there are positive constants K, α, and β such that*

$$\|\mathbf{G}(\cdot,\cdot)\|_\infty \le K\begin{cases} e^{\alpha(\xi-x)}, & \text{if } \xi \le x \\ e^{\beta(x-\xi)}, & \text{if } \xi > x \end{cases}. \tag{7.1.6}$$

The BVP has dichotomy if (7.1.6) holds with $\alpha = \beta = 0$.

Remark 1. The notions of dichotomy and exponential dichotomy correspond to stability and asymptotic stability, respectively, for an IVP

It is relatively easy to show that the *stability constant* κ for Example 7.1.1 has a modest size and that this problem has exponential dichotomy (Problem 2). The problem, once again, is that the IVP is unstable and this renders $\mathbf{Q}$ ill conditioned.

There is a alternative shooting procedure for linear systems that is slightly more efficient than (7.1.2 - 7.1.3) when m is large. We'll present it, although it will not solve our difficulties.

1. Obtain a particular solution $\mathbf{v}$ that satisfies

$$\mathbf{v}' = \mathbf{A}\mathbf{v} + \mathbf{b}, \qquad \mathbf{L}\mathbf{v}(a) = \mathbf{l}. \tag{7.1.7}$$

   We'll have to describe a means of computing $\mathbf{v}(a)$ so that it satisfies the initial condition (7.1.1b), but let's postpone this.

2. Obtain linearly-independent solutions $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(r)}$, to the IVPs

$$(\mathbf{u}^{(i)})' = \mathbf{A}\mathbf{u}^{(i)}, \qquad \mathbf{L}\mathbf{u}^{(i)}(a) = \mathbf{0}, \qquad i = 1, 2, \ldots, r. \tag{7.1.8}$$

   Again, we'll assume that a means of finding $\mathbf{u^{(i)}}$, $i = 1, 2, \ldots, r$, to satisfy the initial condition (7.1.1b) can be found. With this, we observe that the work of solving this system is about half that of the previous procedure (7.1.2c) when the number of terminal conditions r is approximately $m/2$.

3. Write the general solution of the linear BVP (7.1.1) as

$$\mathbf{y}(x) = c_1\mathbf{u}^{(1)}(x) + c_2\mathbf{u}^{(2)}(x) + \ldots + c_r\mathbf{u}^{(r)}(x) + \mathbf{v}(x)$$

   or

$$\mathbf{y}(x) = \mathbf{U}(x)\mathbf{c} + \mathbf{v}(x) \tag{7.1.9a}$$

   where

$$\mathbf{U}(x) = [\mathbf{u}^{(1)}(x),\ \mathbf{u}^{(2)}(x),\ \ldots,\ \mathbf{u}^{(r)}(x)], \qquad \mathbf{c} = [c_1,\ c_2,\ \ldots,\ c_r]^T. \tag{7.1.9b}$$

4. By construction, this representation satisfies the ODE (7.1.1a) and the initial conditions (7.1.1b). It remains to satisfy the terminal conditions (7.1.1c) and this can be done by determining $\mathbf{c}$ so that

$$\mathbf{R}\mathbf{y}(b) = \mathbf{R}[\mathbf{U}(b)\mathbf{c} + \mathbf{v}(b)] = \mathbf{r}$$

or

$$\mathbf{R}\mathbf{U}(b)\mathbf{c} = \mathbf{r} - \mathbf{R}\mathbf{v}(b). \tag{7.1.10}$$

As noted, the main savings of this procedure relative to (7.1.2-7.1.3) is the reduced number of ODEs that have to be solved. This is offset by (possible) difficulties associated with selecting a basis $\mathbf{U}$, $\mathbf{v}$ to satisfy the initial conditions (7.1.1b).

Example 7.1.2. Conte [5] constructed the constant-coefficient fourth-order BVP

$$y^{iv} - (1+R)y'' + Ry = -1 + \frac{Rx^2}{2}, \qquad 0 < x < 1,$$

$$y(0) = y'(0) = 1, \qquad y(1) = \frac{3}{2} + \sinh 1, \qquad y'(1) = 1 + \cosh 1.$$

to try to explain difficulties that researchers were having when solving certain flow-instability problems. We'll rewrite the problem as the first-order system

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -R & 0 & 1+R & 0 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 + Rx^2/2 \end{bmatrix}.$$

$$\mathbf{L} = \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad \mathbf{l} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad \mathbf{r} = \begin{bmatrix} \frac{3}{2} + \sinh 1 \\ 1 + \cosh 1 \end{bmatrix}.$$

Let's go through the steps in solving this problem by the shooting procedure that we just described. We note that

$$\mathbf{v}(0) = [1\ 1\ 0\ 0]^T$$

satisfies the initial conditions and, thus, the particular-solution problem (7.1.7) has been specified. Similarly,

$$\mathbf{u}^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{u}^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

satisfy trivial versions of the initial conditions; hence, the homogeneous problems (7.1.9) are specified.

Conte [5] solved the IVPs (7.1.7, 7.1.9) using a fixed-step Runge-Kutta method. (It was done in the 1960s.) His results for the error $\|e_1\|_\infty$ in the first component $y_1(x)$ of the solution are reported in Table 7.1.2 for $R = 400, 3600$. The exact solution of this problem is

$$y_1(x) = 1 + \frac{x^2}{2} + \sinh x.$$

R	$\|\|e_1\|\|_\infty$
400	0.00173
3600	4000

Table 7.1.2: Maximum errors of Example 7.1.2 using Conte's [5] linear shooting procedure.

The difficulties with this problem are similar to those of Example 7.1.1. The fundamental solutions of the ODE are

$$\sinh x, \qquad \cosh x, \qquad \sinh \sqrt{R}x, \qquad \cosh \sqrt{R}x.$$

Although the exact solution doesn't depend on the rapidly growing components ($\sinh \sqrt{R}x$ and $\cosh \sqrt{R}x$), the IVPs (7.1.7) and (7.1.9) do. Thus, the matrix $\mathbf{RU}(b)$ appearing in (7.1.10) will be ill-conditioned when the parameter R is large. The two components of the fundamental solution $\sinh \sqrt{R}x$ and $\cosh \sqrt{R}x$ are numerically linearly dependent for large values of $\sqrt{R}x$. Although the exact solution of the BVP is independent of these rapidly growing components, small round off errors introduce them and they eventually dominate the exact solution.

Problems

1. Show that the solution representation (7.1.4 satisfies the linear BVP (7.1.1).

2. Find the Green's function for the BVP of Example 7.1.1. Find the stability constant. Show that this problem is stable and has exponential dichotomy.

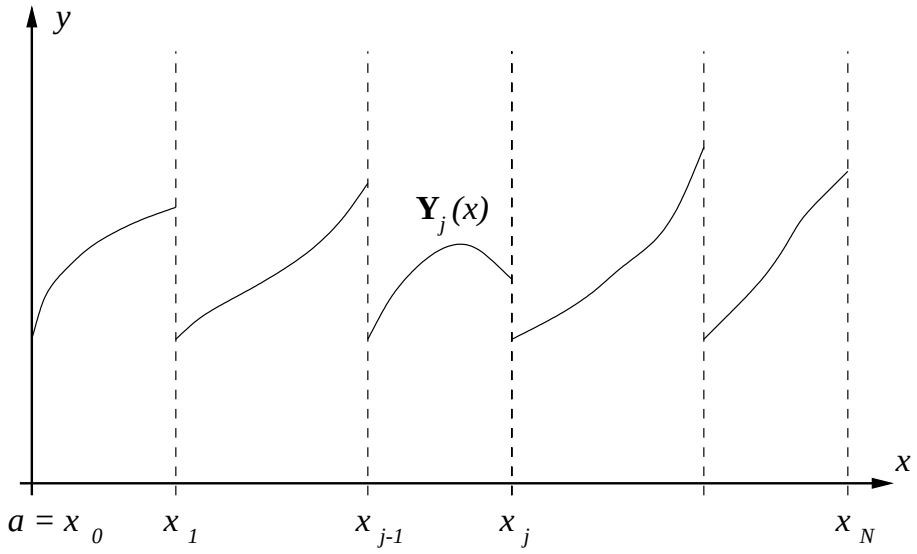

Figure 7.2.1: Geometry and nomenclature for multiple shooting.

7.2 Multiple Shooting

As seen in Section 7.1, errors may accumulate as the solution is generated from the initial to the terminal point. Since round-off errors increase in proportion to the number of operations, they can be reduced by integrating over shorter distances. With this procedure, called *parallel* or *multiple shooting*, we create IVPs that start at several points on $[a, b]$. Let us illustrate the procedure with a linear m-dimensional BVP of the form

$$\mathbf{y}' = \mathbf{A}\mathbf{y} + \mathbf{b}, \qquad a < x < b, \tag{7.2.1a}$$

$$\mathbf{L}\mathbf{y}(a) = \mathbf{l}, \tag{7.2.1b}$$

$$\mathbf{R}\mathbf{y}(b) = \mathbf{r}, \tag{7.2.1c}$$

with $\mathbf{L}$ of dimension $l \times m$ and $\mathbf{R}$ of dimension $r \times m$. Let us further divide $[a, b]$ into N subintervals as shown in Figure 7.2.1. Multiple shooting procedures that are based on the strategies (7.1.2 - 7.1.3) and (7.1.7 - 7.1.10) have been developed. The first approach is the simpler of the two, so let's begin there.

1. On each subinterval $(x_{j-1}, x_j]$, $j = 1, 2, \ldots, N$, solve the IVPs

$$\mathbf{v}_j' = \mathbf{A}\mathbf{v}_j + \mathbf{b}, \qquad \mathbf{v}_j(a) = \mathbf{0}, \tag{7.2.2a}$$

$$\mathbf{Y}_j' = \mathbf{A}\mathbf{Y}_j, \qquad \mathbf{Y}_j(a) = \mathbf{I}, \qquad x_{j-1} < x \le x_j, \qquad j = 1, 2, \ldots, N, \tag{7.2.2b}$$

where $\boldsymbol{\alpha}_j$, $j = 1, 2, \ldots, N$, are chosen and $\mathbf{I}$ is the $m \times m$ identity matrix.

2. As with simple shooting, consider the solution of the BVP in the form (7.1.2a), which now becomes

$$\mathbf{y}(x) = \mathbf{Y}_j(x)\mathbf{c}_j + \mathbf{v}_j(x), \qquad x_{j-1} < x \le x_j, \qquad j = 1, 2, \ldots, N. \tag{7.2.3}$$

The solution must be continuous at the interior shooting points x_j, $j = 1, 2, \ldots, N-1$; thus, with the initial conditions specified by (7.2.2)

$$\mathbf{Y}_j(x_j)\mathbf{c}_j + \mathbf{v}_j(x_j) = \mathbf{c}_{j+1}, \qquad j = 1, 2, \ldots, N-1.$$

The boundary conditions (7.2.1b,c) must also be satisfied and, with (7.2.3) and (7.2.2), this implies

$$\mathbf{L}\mathbf{y}(a) = \mathbf{L}\mathbf{Y}_1\mathbf{c}_1 + \mathbf{L}\mathbf{v}(a) = \mathbf{L}\mathbf{c}_1 = \mathbf{l}, \qquad \mathbf{R}\mathbf{y}(b) = \mathbf{R}\mathbf{Y}_N(b)\mathbf{c}_N + \mathbf{R}\mathbf{v}_N(b) = \mathbf{r}.$$

Writing this system in matrix form

$$\mathbf{A}\mathbf{c} = \mathbf{g} \tag{7.2.4a}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{L} & & & & & \\ -\mathbf{Y}_1(x_1) & \mathbf{I} & & & & \\ & -\mathbf{Y}_2(x_2) & \mathbf{I} & & & \\ & & \ddots & \ddots & & \\ & & & -\mathbf{Y}_{N-1}(x_{N-1}) & \mathbf{I} & \\ & & & & \mathbf{R}\mathbf{Y}_N(b) & \end{bmatrix} \tag{7.2.4b}$$

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_N \end{bmatrix}, \qquad \mathbf{g} = \begin{bmatrix} \mathbf{l} \\ \mathbf{v}_1(x_1) \\ \vdots \\ \mathbf{v}_{N-1}(x_{N-1}) \\ \mathbf{r} - \mathbf{R}\mathbf{v}_N(b) \end{bmatrix}. \tag{7.2.4c}$$

We will encounter algebraic systems like this in this and the following chapters. We'll describe procedures for solving them in Chapter 8 that only require $O(N)$ operations.

3. Once $\mathbf{c}$ has been determined, the BVP solution may be reconstructed from (7.2.3).

Remark 1. All of the IVPs (7.2.2) can be solved in parallel on a computer with N processors. This is why multiple shooting is also called parallel shooting.

Conte [5] described a method for stabilizing the procedure (7.1.7 - 7.1.10) which was later automated and improved by Scott and Watts [11]. The technique is less parallel than the one that we just described, but it does offer some advantages. To begin, we calculate solutions of the IVPs

$$\mathbf{U}_1' = \mathbf{A}\mathbf{U}_1, \qquad x_0 < x < x_1, \qquad \mathbf{L}\mathbf{U}_1(a) = 0,$$

$$\mathbf{v}_1' = \mathbf{A}\mathbf{v}_1 + \mathbf{b}, \qquad x_0 < x \leq x_1, \qquad \mathbf{L}\mathbf{v}_1(a) = \mathbf{l},$$

and write the solution of the BVP as

$$\mathbf{y}(x) = \mathbf{U}_1(x)\mathbf{c}_1 + \mathbf{v}_1(x), \qquad x_0 < x < x_1.$$

As in Section 7.1,

$$\mathbf{U}_1 = [\mathbf{u}_1^{(1)},\ \mathbf{u}_1^{(2)},\ \ldots,\ \mathbf{u}_1^{(r)}].$$

The integration proceeds to a point x_1 where, due to the accumulation of round off errors, the columns of $\mathbf{U}_1$ no longer form a good linearly independent basis for the solution. The point x_1 is determined as part of the solution process but, for the present, let's assume that it is known.

New initial conditions $\mathbf{U}_2(x_1)$ are determined by orthogonalizing the columns of $\mathbf{U}_1(x_1)$. We'll subsequently show that this is equivalent to finding an upper triangular matrix $\mathbf{P}_1$ such that

$$\mathbf{U}_2(x_1)\mathbf{P}_1 = \mathbf{U}_1(x_1).$$

We also select new initial conditions $\mathbf{v}_2(x_1)$ for the particular solution that are in the orthogonal complement of $\mathbf{U}_2(x_1)$. Again, we'll postpone the details and note that this is equivalent to determining a vector $\mathbf{w}_1$ such that

$$\mathbf{v}_2(x_1) = \mathbf{v}_1(x_1) - \mathbf{U}_2(x_1)\mathbf{w}_1.$$

The time integration can now continue by solving

$$\mathbf{U}_2' = \mathbf{A}\mathbf{U}_2, \qquad x_1 < x < x_2,$$

$$\mathbf{v}_2' = \mathbf{A}\mathbf{v}_2 + \mathbf{b}, \qquad x_1 < x < x_2.$$

The quantities $\mathbf{U}_2(x_1)$ and $\mathbf{v}_2(x_1)$ are used as initial conditions. The process continues to x_2 where the columns of $\mathbf{U}_2(x_2)$ fail to satisfy a linear independence test.

The general procedure is summarized as follows.

1. Determine $\mathbf{U}_1(a)$ and $\mathbf{v}_1(a)$ as solutions of

$$\mathbf{L}\mathbf{U}_1(a) = \mathbf{0}, \tag{7.2.5a}$$

$$\mathbf{L}\mathbf{v}_1(a) = \mathbf{l}. \tag{7.2.5b}$$

2. Beginning with $j = 1$, solve the IVPs

$$\mathbf{U}_j' = \mathbf{A}\mathbf{U}_j, \qquad x_{j-1} < x < x_j, \tag{7.2.6a}$$

$$\mathbf{v}_j' = \mathbf{A}\mathbf{v}_j + \mathbf{b}, \qquad x_{j-1} < x < x_j. \tag{7.2.6b}$$

   Let the solution of the BVP have the form

$$\mathbf{y}(x) = \mathbf{U}_j(x)\mathbf{c}_j + \mathbf{v}_j(x), \qquad x_{j-1} < x < x_j. \tag{7.2.6c}$$

3. At a failure of a linear independence test, compute new initial conditions $\mathbf{U}_{j+1}(x_j)$ and $\mathbf{v}_{j+1}(x_j)$ by solving

$$\mathbf{U}_{j+1}(x_j)\mathbf{P}_j = \mathbf{U}_j(x_j) \tag{7.2.7a}$$

$$\mathbf{v}_{j+1}(x_j) = \mathbf{v}_j(x_j) + \mathbf{U}_{j+1}(x_j)\mathbf{w_j}. \tag{7.2.7b}$$

4. Repeat the previous two steps until reaching the terminal point $x_N = b$. The BVP solution at b is

$$\mathbf{y}(b) = \mathbf{U}_N(b)\mathbf{c}_N + \mathbf{v}_N(b).$$

5. Using the terminal condition

$$\mathbf{R}\mathbf{y}(b) = \mathbf{R}\mathbf{U}_N(b)\mathbf{c}_N + \mathbf{R}\mathbf{v}_N(b) = \mathbf{r},$$

   determine $\mathbf{c}_N$ as the solution of

$$\mathbf{R}\mathbf{U}_N(b)\mathbf{c}_N = \mathbf{R}\mathbf{v}_N(b) - \mathbf{r} \tag{7.2.8}$$

6. Once $\mathbf{c}_N$ has been determined, the remaining $\mathbf{c}_j$, $j = 1, 2, \ldots, N-1$, are determined by enforcing solution continuity at the orthogonalization points, i.e.,

$$\mathbf{U}_j(x_j)\mathbf{c}_j + \mathbf{v}_j(x_j) = \mathbf{U}_{j+1}(x_j)\mathbf{c}_{j+1} + \mathbf{v}_{j+1}(x_j).$$

This relation can be simplified by the use of (7.2.7) to

$$\mathbf{U}_{j+1}(x_j)\mathbf{P}_j\mathbf{c}_j + \mathbf{v}_j(x_j) = \mathbf{U}_{j+1}(x_j)\mathbf{c}_{j+1} + \mathbf{v}_j(x_j) + \mathbf{U}_{j+1}(x_j)\mathbf{w}_j$$

or

$$\mathbf{U}_{j+1}(x_j)[\mathbf{P}_j\mathbf{c}_j - \mathbf{c}_{j+1} - \mathbf{w}_j] = 0.$$

Since $\mathbf{P}_j$ is non-singular, we may take the solution of this system as

$$\mathbf{P}_j\mathbf{c}_j = \mathbf{c}_{j+1} + \mathbf{w}_j, \qquad j = N-1, N-2, \ldots, 1. \tag{7.2.9}$$

Several details still remain, including

1. the satisfaction of the initial conditions (7.2.5a,b),
2. the orthogonalization procedure (7.2.7a,b), and
3. the test for a linearly independent basis.

Consider the solution of the under-determined system (7.2.5a,b) first. The r columns of $\mathbf{U}_1(a)$ are calculated so that they are mutually orthogonal and span the null space of $\mathbf{L}$. A procedure follows.

1. Reduce $\mathbf{L}^T$ to upper triangular form using the QR algorithm ([6], Chapter 3). This involves finding an $m \times m$ orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{Q}\mathbf{L}^T = \begin{bmatrix} \mathbf{T} \\ \mathbf{0} \end{bmatrix}. \tag{7.2.10a}$$

Recall that an orthogonal matrix is one where

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}. \tag{7.2.10b}$$

The matrix $\mathbf{T}$ is an $l \times l$ upper triangular matrix and the zero matrix is $r \times l$. Pivoting can frequently be ignored with orthogonal transformations and we'll assume this to be the case here.

2. Choose

$$\mathbf{U}_1(a) = \mathbf{Q}^T \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \tag{7.2.11}$$

where $\mathbf{I}$ is the $r \times r$ identity matrix and the zero matrix is $l \times r$. Thus $\mathbf{U}_1$ is the last r columns of $\mathbf{Q}^T$. Using (7.2.10) and (7.2.11) we verify that

$$\mathbf{L}\mathbf{U}_1(a) = \mathbf{L}\mathbf{Q}^T \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} = [\mathbf{T}^T \ \mathbf{0}] \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} = \mathbf{0}.$$

3. The vector $\mathbf{v}_1(a)$ is obtained as the least squares solution [9] of (7.2.5b) by solving

$$\mathbf{T}^T\mathbf{s} = \mathbf{l} \tag{7.2.12a}$$

for $\mathbf{s}$ using forward substitution and setting

$$\mathbf{v}_1(a) = \mathbf{Q}^T \begin{bmatrix} \mathbf{s} \\ \mathbf{0} \end{bmatrix}. \tag{7.2.12b}$$

We readily verify that

$$\mathbf{L}\mathbf{v}_1(a) = \mathbf{L}\mathbf{Q}^T \begin{bmatrix} \mathbf{s} \\ \mathbf{0} \end{bmatrix} = [\mathbf{T}^T \ \mathbf{0}] \begin{bmatrix} \mathbf{s} \\ \mathbf{0} \end{bmatrix} = \mathbf{T}^T\mathbf{s} = \mathbf{l}$$

and, using (7.2.12b) and (7.2.11),

$$\mathbf{v}_1^T(a)\mathbf{U}_1(a) = [\mathbf{s}^T \ \mathbf{0}]\mathbf{Q}\mathbf{Q}^T \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} = \mathbf{0}.$$

The orthogonalization task (7.2.7) can also be done by the QR procedure.

1. Determine an orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{Q}\mathbf{U}_j = \begin{bmatrix} \mathbf{P}_j \\ \mathbf{0} \end{bmatrix} \tag{7.2.13a}$$

where $\mathbf{P}_j$ is an $r \times r$ upper triangular matrix and the zero matrix is $l \times r$. (Spatial arguments in (7.2.7) at x_j have been omitted for clarity. Although the symbol $\mathbf{Q}$ has been used in (7.2.10 - 7.2.12), the matrix in (7.2.13a) is generally not the same.)

2. Select

$$\mathbf{U}_{j+1} = \mathbf{Q}^T \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}. \tag{7.2.13b}$$

Using (7.2.13a)

$$\mathbf{U}_j = \mathbf{Q}^T \begin{bmatrix} \mathbf{P}_j \\ \mathbf{0} \end{bmatrix}.$$

According to (7.2.13b)

$$\mathbf{Q}^T = [\mathbf{U}_{j+1}\ \mathbf{V}_{j+1}]$$

Thus,

$$\mathbf{U}_j = [\mathbf{U}_{j+1},\ \mathbf{V}_{j+1}] \begin{bmatrix} \mathbf{P}_j \\ \mathbf{0} \end{bmatrix} = \mathbf{U}_{j+1}\mathbf{P}_j.$$

3. In order to satisfy

$$\mathbf{v}_{j+1}^T \mathbf{U}_{j+1} = \mathbf{0},$$

choose

$$\mathbf{w}_j^T = -\mathbf{v}_j^T \mathbf{U}_{j+1}. \tag{7.2.14}$$

To verify this, consider (7.2.7b), (7.2.14), and

$$\mathbf{v}_{j+1}^T \mathbf{U}_{j+1} = (\mathbf{v}_j^T + \mathbf{w}_j^T \mathbf{U}_{j+1}^T)\mathbf{U}_{j+1} = \mathbf{v}_j^T \mathbf{U}_{j+1} + \mathbf{w}_j^T = \mathbf{0}.$$

Conte [5] suggested examining the "angle" between pairs of columns of $\mathbf{U}_j(x_j)$ and re-orthogonalizing whenever this angle became too small. Again, we'll omit the spatial argument x_j and compute

$$\cos\alpha_{ik} = \frac{(\mathbf{u}_j^{(i)}, \mathbf{u}_j^{(k)})}{|\mathbf{u}_j^{(i)}|_2 |\mathbf{u}_j^{(k)}|_2}, \qquad i,k = 1,2,\ldots,r-l, \tag{7.2.15a}$$

where

$$(\mathbf{u},\mathbf{w}) = \mathbf{v}^T\mathbf{w}, \qquad |\mathbf{u}|_2 = \sqrt{(\mathbf{u},\mathbf{u})}. \tag{7.2.15b}$$

The matrix $\mathbf{U}_j$ is re-orthogonalized whenever

$$\min_{1\le i,k\le r-l} |\alpha_{ik}| < \epsilon \tag{7.2.15c}$$

Example 7.2.1. Solve Example 7.1.2 with $R = 3600$ using the multiple shooting procedure with orthogonalization described above with $\epsilon = 10°$. The result for the maximum error in the component y_1 of the solution is 2.0×10^{-7}. Four orthogonalization points were used during the integration.

Let us conclude this section with a brief discussion of the QR algorithm to reduce an $m \times n$ $(m \geq n)$ matrix $\mathbf{A}$ to upper triangular form using orthogonal transformations, i.e., for finding an $m \times m$ orthogonal matrix $\mathbf{Q}$ such that

$$\mathbf{QA} = \begin{bmatrix} \mathbf{T} \\ \mathbf{0} \end{bmatrix}, \tag{7.2.16}$$

where $\mathbf{T}$ is $n \times n$.

The procedure can be done in many ways, but we'll focus on the use of *plane reflections* or *Householder transformations*

$$\mathbf{H}(\boldsymbol{\omega}) = \mathbf{I} - 2\frac{\boldsymbol{\omega}\boldsymbol{\omega}^T}{\boldsymbol{\omega}^T\boldsymbol{\omega}}. \tag{7.2.17}$$

Householder transformations satisfy:

1. For all real values of $\alpha \neq 0$,
$$\mathbf{H}(\alpha\boldsymbol{\omega}) = \mathbf{H}(\boldsymbol{\omega}).$$

2. $\mathbf{H}$ is symmetric.

3. $\mathbf{H}$ is orthogonal as we easily check
$$\mathbf{H}^T\mathbf{H} = \mathbf{H}^2 = (\mathbf{I} - 2\frac{\boldsymbol{\omega}\boldsymbol{\omega}^T}{\boldsymbol{\omega}^T\boldsymbol{\omega}})(\mathbf{I} - 2\frac{\boldsymbol{\omega}\boldsymbol{\omega}^T}{\boldsymbol{\omega}^T\boldsymbol{\omega}})$$
or
$$\mathbf{H}^T\mathbf{H} = \mathbf{I} - 4\frac{\boldsymbol{\omega}\boldsymbol{\omega}^T}{\boldsymbol{\omega}^T\boldsymbol{\omega}} + 4\frac{\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{\omega}\boldsymbol{\omega}^T}{(\boldsymbol{\omega}^T\boldsymbol{\omega})^2} = \mathbf{I}.$$

4. $\mathbf{H}(\boldsymbol{\omega})\mathbf{v} = \mathbf{v}$, if and only if $\mathbf{v}^T\boldsymbol{\omega} = \mathbf{0}$.

5. $\mathbf{H}(\boldsymbol{\omega})\boldsymbol{\omega} = -\boldsymbol{\omega}$.

6. Let $\mathbf{u} = \mathbf{v} + \boldsymbol{\omega}$ and $\mathbf{v}^T\boldsymbol{\omega} = \mathbf{0}$, then
$$\mathbf{H}(\boldsymbol{\omega})(\mathbf{v} + \boldsymbol{\omega}) = \mathbf{v} - \boldsymbol{\omega}.$$
Thus, the Householder transformation reflects $\boldsymbol{\omega} + \mathbf{v}$ in a plane through the origin perpendicular to $\boldsymbol{\omega}$.

In order to perform the QR reduction (7.2.16), let $\mathbf{a}_1$ be the first column of $\mathbf{A}$ and choose

$$\boldsymbol{\omega} = \mathbf{a}_1 \pm \lambda_1 \mathbf{e}_1$$

where $\mathbf{e}_1$ is the first column of the identity matrix and

$$\lambda_1 = \sqrt{\mathbf{a}_1^T \mathbf{a}_1}.$$

Letting $\mathbf{H}^{(1)} = \mathbf{H}(\boldsymbol{\omega})$, we have

$$\mathbf{H}^{(1)}\mathbf{A} = \begin{bmatrix} \pm\lambda_1 & \times & & \times \\ 0 & \times & & \times \\ \vdots & & \ddots & \\ 0 & \times & & \times \end{bmatrix} = \begin{bmatrix} \pm\lambda_1 & (\mathbf{v}^{(1)})^T \\ \mathbf{0} & \mathbf{A}^{(1)} \end{bmatrix} = \mathbf{A}^{(1)}.$$

The same process is repeated on the $(m-1)\times(n-1)$ portion of $\mathbf{A}$. Thus, if $\mathbf{a}_1^{(1)}$ is the first column of $\mathbf{A}^{(1)}$, choose

$$\boldsymbol{\omega}^{(1)} = \mathbf{a}_1^{(1)} \pm \lambda_2 \mathbf{e}_1^{(1)}$$

where $\mathbf{e}_1^{(1)}$ is the first column of the $(m-1)\times(m-1)$ identity matrix. The transformation $H^{(2)}$ is

$$\mathbf{H}^{(2)} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}(\boldsymbol{\omega}^{(1)}) \end{bmatrix}$$

and the result after an application of $\mathbf{H}^{(2)}$ is

$$\mathbf{H}^{(2)}\mathbf{A}^{(1)} = \begin{bmatrix} \times & \times & \times & & \times \\ 0 & \times & \times & & \times \\ 0 & 0 & \times & & \times \\ \vdots & & & \ddots & \\ 0 & 0 & \times & & \times \end{bmatrix}.$$

After $p = \min(m-1, n)$ steps, we have

$$\mathbf{H}^{(p)}\mathbf{H}^{(p-1)}\ldots\mathbf{H}^{(1)}\mathbf{A} = \begin{bmatrix} \mathbf{T} \\ \mathbf{0} \end{bmatrix};$$

thus,

$$\mathbf{Q} = \mathbf{H}^{(p)}\mathbf{H}^{(p-1)}\ldots\mathbf{H}^{(1)}$$

and

$$\mathbf{H}^{(k)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}(\boldsymbol{\omega}^{(k-1)}) \end{bmatrix},$$

where the identity matrix is $(k-1)\times(k-1)$.

At each stage of the procedure, the sign of λ_k is chosen so that cancellation is avoided. Thus, if $a_{11} \geq 0$ choose the positive sign for λ_1, and if $a_{11} < 0$ choose the negative sign. The reduction can be done efficiently in about $2n^2(m-n/3)$ multiplications plus additions and approximately n square roots. Techniques to save storage and avoid explicit multiplication to determine $\mathbf{Q}$ are described by Demmel [6].

7.3 Multiple Shooting for Nonlinear Problems

We extend the multiple shooting procedures of Section 7.2 to nonlinear problems

$$\mathbf{y}' = \mathbf{f}(x,\mathbf{y}), \qquad a < x < b, \qquad \mathbf{g}_L(\mathbf{y}(a)) = 0, \qquad \mathbf{g}_R(\mathbf{y}(b)) = 0, \tag{7.3.1}$$

by linearization in function space. Thus, we assume $\bar{\mathbf{y}}$ to be known and let

$$\mathbf{y}(x) = \bar{\mathbf{y}}(x) + \boldsymbol{\delta}\mathbf{y}(x) \tag{7.3.2}$$

and use a Taylor's series expansion of $\mathbf{f}$ to obtain

$$\mathbf{f}(x, \bar{\mathbf{y}} + \boldsymbol{\delta}\mathbf{y}) = \mathbf{f}(x,\bar{\mathbf{y}}) + \mathbf{f}_{\mathbf{y}}(x,\bar{\mathbf{y}})\boldsymbol{\delta}\mathbf{y} + O(\boldsymbol{\delta}\mathbf{y}^2).$$

Substituting these results into (7.3.1) and neglecting the $O(\boldsymbol{\delta}\mathbf{y}^2)$ terms

$$\bar{\mathbf{y}}' + \boldsymbol{\delta}\mathbf{y}' = \mathbf{f}_{\mathbf{y}}(x,\bar{\mathbf{y}})\boldsymbol{\delta}\mathbf{y} + \mathbf{f}(x,\bar{\mathbf{y}})$$

or

$$\boldsymbol{\delta}\mathbf{y}' = \mathbf{f}_{\mathbf{y}}(x,\bar{\mathbf{y}})\boldsymbol{\delta}\mathbf{y} + \mathbf{f}(x,\bar{\mathbf{y}}) - \bar{\mathbf{y}}'.$$

This has the form of a linear system

$$\boldsymbol{\delta}\mathbf{y}' = \mathbf{A}(x)\boldsymbol{\delta}\mathbf{y} + \mathbf{b}(x) \tag{7.3.3a}$$

with

$$\mathbf{A}(x) = \mathbf{f}_{\mathbf{y}}(x,\bar{\mathbf{y}}), \qquad \mathbf{b}(x) = \mathbf{f}(x,\bar{\mathbf{y}}) - \bar{\mathbf{y}}'. \tag{7.3.3b}$$

Linearizing the left boundary condition yields

$$\mathbf{g}_L(\bar{\mathbf{y}}(a) + \boldsymbol{\delta}\mathbf{y}(a)) = \mathbf{g}_L(\bar{\mathbf{y}}(a)) + \mathbf{g}_{L_{\mathbf{y}}}(\bar{\mathbf{y}}(a))\boldsymbol{\delta}\mathbf{y}(a) = \mathbf{0}.$$

This has the form of the linear boundary condition

$$\mathbf{L}\boldsymbol{\delta}\mathbf{y}(a) = \mathbf{l} \tag{7.3.4a}$$

with

$$\mathbf{L} = \mathbf{g}_{L_\mathbf{y}}(\bar{\mathbf{y}}(a)), \qquad \mathbf{l} = -\mathbf{g}_L(\bar{\mathbf{y}}(a)). \tag{7.3.4b}$$

At the right end, we have

$$\mathbf{R}\boldsymbol{\delta}\mathbf{y}(b) = \mathbf{r} \tag{7.3.5a}$$

with

$$\mathbf{R} = \mathbf{g}_{R_\mathbf{y}}(\bar{\mathbf{y}}(b)), \qquad \mathbf{r} = -\mathbf{g}_R(\bar{\mathbf{y}}(b)). \tag{7.3.5b}$$

The linearized system (7.3.3 - 7.3.5) may be solved by iteration using the procedures of Section 7.2; thus,

1. Beginning with an initial guess $\mathbf{y}^{(0)}(x)$, $0 \le x \le 1$,

2. Solve the linear system

$$\boldsymbol{\delta}\mathbf{y}'^{(\nu)} = \mathbf{A}^{(\nu)}(x)\boldsymbol{\delta}\mathbf{y}^{(\nu)} + \mathbf{b}^{(\nu)}(x)$$

$$\mathbf{L}^{(\nu)}\boldsymbol{\delta}\mathbf{y}^{(\nu)}(a) = \mathbf{l}^{(\nu)}, \qquad \mathbf{R}^{(\nu)}\boldsymbol{\delta}\mathbf{y}^{(\nu)}(b) = \mathbf{r}^{(\nu)},$$

   where $\mathbf{A}^{(\nu)}(x) = \mathbf{f}_\mathbf{y}(x, \mathbf{Y}^{(\nu)})$, *etc.*

3. After each iteration, set

$$\mathbf{y}^{(\nu+1)} = \mathbf{y}^{(\nu)} + \boldsymbol{\delta}\mathbf{y}^{(\nu)}$$

   and repeat the procedure until convergence.

The procedure is awkward since interpolation of $\mathbf{y}^{(\nu)}$ will generally be required to determine $\mathbf{A}^{(\nu)}(x)$, *etc.* at locations that are needed by a variable-step IVP procedure.

Bibliography

[1] U.M. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations.* SIAM, Philadelphia, second edition, 1995.

[2] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[3] G. Birkhoff and G.-C. Rota. *Ordinary Differential Equations.* John Wiley and Sons, New York, third edition, 1978.

[4] W.E. Boyce and R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* John Wiley and Sons, New York, third edition, 1977.

[5] S.D. Conte. The numerical solution of linear boundary value problems. *SIAM Review*, 8:309–321, 1975.

[6] J.W. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[7] J.C. Diaz, G. Fairweather, and P. Keast. Fortran packages for solving almost block diagonal linear systems by modified alternate row and column elimination. *ACM Transactions on Mathematical Software*, 9:358–375, 1981.

[8] J.E. Flaherty and W. Mathon. Collocation with polynomial and tension splines for singularly perturbed boundary value problems. *SIAM J. Sci. Stat. Comput*, 1:260–289, 1980.

[9] G. Golub. Numerical methods for solving linear least squares problems. *Numer. Math.*, 7:206–216, 1965.

[10] H.B. Keller. Accurate difference methods for nonlinear two-point boundary value problems. *SIAM J. Numer. Anal.*, 11:305–320, 1974.

[11] M.R. Scott and H.A. Watts. Computational solution of linear two point boundary value problems via orthonormalization. *SIAM J. Numer. Anal.*, 14:40–70, 1977.

[12] J.M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.*, 13:71–75, 1976.

Chapter 8

Finite Difference Methods

8.1 Introduction

Let's continue the discussion of finite difference procedures for two-point boundary value problems that we began in Section 6.3 with a brief review of the basic theoretical concepts. Recall that we had examined the solution of second-order linear boundary value problems of the form

$$\mathcal{L}y = y'' + p(x)y' + q(x)y - r(x) = 0, \qquad a < x < b, \tag{8.1.1a}$$

$$y(a) = A, \qquad y(b) = B. \tag{8.1.1b}$$

After introducing a uniform mesh $x_i = a + ih$, $i = 0, 1, \ldots, N$, with $h = (b-a)/N$, we approximated the derivatives in (8.1.1a) by central differences and solved the discrete problem

$$\begin{aligned} \mathcal{L}_h y_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p(x_i)\frac{y_{i+1} - y_{i-1}}{2h} \quad &+ \quad q(x_i)y_i - r(x_i) = 0, \\ i \quad &= \quad 1, 2, \ldots, N-1, \end{aligned} \tag{8.1.2a}$$

$$y_0 = A, \qquad y_N = B. \tag{8.1.2b}$$

The discrete solution y_i is an approximation of $y(x_i)$, $i = 0, 1, \ldots, N$.

In Example 6.3.1, we saw that the local discretization error of (8.1.2) at x_i is (Definition 6.3.1)

$$\tau_i = \mathcal{L}_h y(x_i) = \frac{h^2}{12} y^{iv}(\xi_i) + p(x_i)\frac{h^2}{6} y'''(\eta_i), \qquad \xi_i, \eta_i \in (x_{i-1}, x_i). \tag{8.1.3}$$

With this, we can define consistency in an analogous fashion to that of an IVP.

Definition 8.1.1. *The difference scheme (8.1.2) is consistent with the BVP (8.1.1) if*

$$\lim_{h\to 0} \tau_i = 0.$$

As with IVPs, we also need the basic notions of convergence and stability.

Definition 8.1.2. *The difference scheme (8.1.2) is convergent at a point x_i if*

$$\lim_{h\to 0, N\to\infty} |y_i - y(x_i)| = 0,$$

such that the location of x_i remains fixed as $i \to \infty$ and $h \to 0$.

Recall that stability of the BVP implies the existence of a constant $\kappa > 0$ such that

$$\|y(\cdot)\|_\infty \le \kappa[|A| + |B| + \int_a^b |r(x)|dx]. \tag{8.1.4}$$

Thus, the BVP is bounded by its data. One often restricts κ to be of "moderate size" which means that it shouldn't be much bigger than unity. Stability of the discrete system (8.1.2) follows in analogous fashion.

Definition 8.1.3. *The difference equation (8.1.2) is stable if there exists a constant $C > 0$ such that*

$$|\mathbf{y}|_\infty \le \bar{\kappa}[|y_0| + |y_N| + |\mathbf{r}|_\infty\} \tag{8.1.5a}$$

as $h \to 0$.

The norm in (8.1.5a) may be taken as the discrete maximum norm (7.1.5b)

$$|\mathbf{y}|_\infty = \max_{0\le i\le N} |y_i|. \tag{8.1.5b}$$

With the global error defined as

$$e_i = y(x_i) - y_i, \qquad i = 0, 1, \ldots, N, \tag{8.1.6}$$

we may use (8.1.1 - 8.1.3) to show

$$\mathcal{L}_h e_i = \mathcal{L}_h[y(x_i) - y_i] = \tau_i, \qquad i = 1, 2, \ldots, N-1, \tag{8.1.7a}$$

$$e_0 = e_N = 0. \tag{8.1.7b}$$

Thus, for linear problems, the global error satisfies the difference equation (8.1.2) with the local discretization error as an inhomogeneous term.

It is actually not difficult to prove that the difference equation (8.1.2) satisfies a stability bound of the form (8.1.5), but we will postpone this and, assuming that it's true, obtain the following result.

Theorem 8.1.1. *Assume that the solution of (8.1.1) has a bounded fourth derivative and that the solution of (8.1.2) satisfies the stability condition (8.1.5), then the difference scheme (8.1.2) is convergent for* $i = 0, 1, \ldots, N$.

Proof. The assumptions are sufficient to apply (8.1.5a) to the global error and obtain

$$|\mathbf{e}|_\infty \leq \bar{\kappa} |\boldsymbol{\tau}|_\infty \leq Ch^2.$$

Thus, the difference solution converges as $h \to 0$. □

Much stronger results are possible and will be considered subsequently. For the moment note that, in analogy with our work on IVPs, we have stability and consistency implying convergence. However, in contrast, the local and global discretization errors converge at the same rates.

8.2 The Box Scheme for First-Order Systems

We'll extend finite difference methods to nonlinear first-order BVPs of the form

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \qquad a < x < b, \tag{8.2.1a}$$

$$\mathbf{g}_L(\mathbf{y}(a)) = \mathbf{g}_R(\mathbf{y}(b)) = 0. \tag{8.2.1b}$$

As in Chapter 7, $\mathbf{y}$ and $\mathbf{f}$ are m-vectors, $\mathbf{g}_L$ is an l-vector, and $\mathbf{g}_R$ is an r-vector with $l + r = m$.

We will discretize (8.2.1) by a variant of the trapezoidal rule called the "box scheme" that was proposed by Keller [6] and developed by Lentini and Pereyra [7]. Towards this end, introduce a partition of (a, b) into N subintervals as shown in Figure 8.2.1. Let

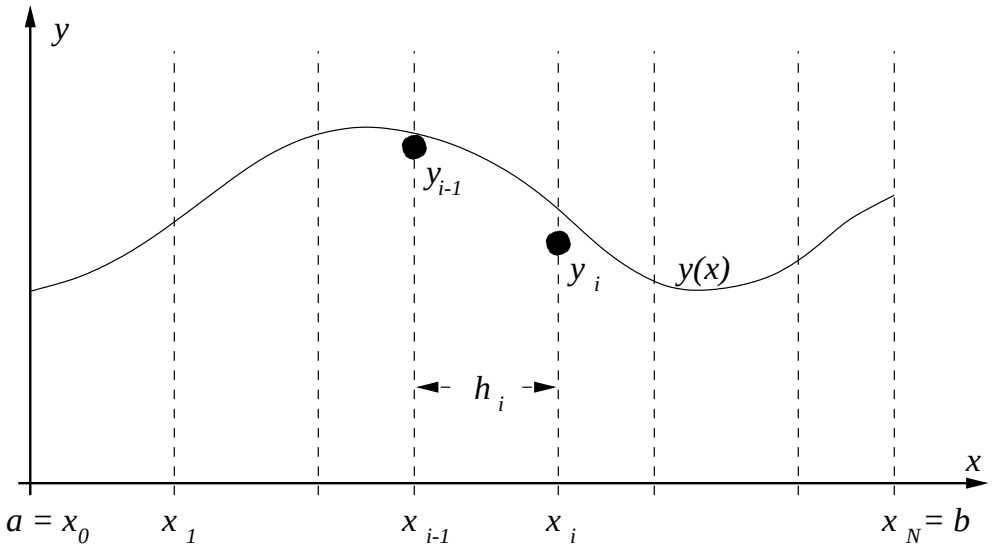

Figure 8.2.1: Mesh nomenclature for the box scheme.

$$h_i = x_i - x_{i-1} \tag{8.2.2}$$

and approximate (8.2.1a) using the trapezoidal rule

$$\boldsymbol{\phi}_i(\mathbf{y}) = \frac{\mathbf{y}_i - \mathbf{y}_{i-1}}{h_i} - \frac{\mathbf{f}(x_i, \mathbf{y}_i) + \mathbf{f}(x_{i-1}, \mathbf{y}_{i-1})}{2} = \mathbf{0}, \qquad i = 1, 2, \ldots, N, \tag{8.2.3a}$$

where

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix}. \tag{8.2.3b}$$

The boundary conditions (8.2.1b) complete the specification of the discrete problem as

$$\mathbf{g}_L(\mathbf{y}_0) = \mathbf{g}_R(\mathbf{y}_N) = \mathbf{0}. \tag{8.2.3c}$$

The BVP has been approximated by a nonlinear algebraic system (8.2.3) of dimension mN for the N vector of unknowns $\mathbf{y}_i$, $i = 0, 1, \ldots, N$.

The midpoint rule can be used in place of the trapezoidal rule to obtain

$$\boldsymbol{\phi}_i(\mathbf{y}) = \frac{\mathbf{y}_i - \mathbf{y}_{i-1}}{h_i} - \mathbf{f}(x_{i-1/2}, \frac{\mathbf{y}_i + \mathbf{y}_{i-1}}{2}) = 0. \tag{8.2.4}$$

The midpoint and trapezoidal rules have similar, but not identical, accuracy and stability properties.

Focusing on the trapezoidal rule, we'll linearize (8.2.3) by Newton's method. For (8.2.3a) we have

$$\frac{\partial \boldsymbol{\phi}_i(\mathbf{y}^{(\nu)})}{\partial \mathbf{y}} \boldsymbol{\Delta}\mathbf{y}^{(\nu)} = -\boldsymbol{\phi}_i(\mathbf{y}^{(\nu)}), \qquad i = 1, 2, \ldots, N, \tag{8.2.5a}$$

and for (8.2.3b) we have

$$\frac{\partial \mathbf{g}_L(\mathbf{y}_0^{(\nu)})}{\partial \mathbf{y}_0} \boldsymbol{\Delta}\mathbf{y}_0^{(\nu)} = -\mathbf{g}_L(\mathbf{y}_0^{(\nu)}), \tag{8.2.5b}$$

$$\frac{\partial \mathbf{g}_R(\mathbf{y}_N^{(\nu)})}{\partial \mathbf{y}_N} \boldsymbol{\Delta}\mathbf{y}_N^{(\nu)} = -\mathbf{g}_R(\mathbf{y}_N^{(\nu)}), \tag{8.2.5c}$$

with

$$\mathbf{y}^{(\nu+1)} = \mathbf{y}^{(\nu)} + \boldsymbol{\Delta}\mathbf{y}^{(\nu)}. \tag{8.2.5d}$$

Using (8.2.3a), the two nonzero contributions to the Jacobian in (8.2.5a) are

$$\frac{\partial \boldsymbol{\phi}_i(\mathbf{y}^{(\nu)})}{\partial \mathbf{y}_{i-1}} := \mathbf{L}_i^{(\nu)} = -\frac{\mathbf{I}}{h_i} - \frac{\mathbf{f}_{\mathbf{y}}(x_{i-1}, \mathbf{y}_{i-1}^{(\nu)})}{2} \tag{8.2.6a}$$

and

$$\frac{\partial \boldsymbol{\phi}_i(\mathbf{y}^{(\nu)})}{\partial \mathbf{y}_i} := \mathbf{R}_i^{(\nu)} = \frac{\mathbf{I}}{h_i} - \frac{\mathbf{f}_{\mathbf{y}}(x_i, \mathbf{y}_i^{(\nu)})}{2} \tag{8.2.6b}$$

where $\mathbf{I}$ is the $m \times m$ identity matrix.

For consistency and simplicity, we'll also write the Jacobians in (8.2.5b,c) as

$$\mathbf{L}_{\mathbf{0}}^{(\nu)} := \frac{\partial \mathbf{g}_{\mathbf{L}}(\mathbf{y}_{\mathbf{0}}^{(\nu)})}{\partial \mathbf{y}_{\mathbf{0}}} \tag{8.2.6c}$$

$$\mathbf{R}_{\mathbf{N+1}}^{(\nu)} := \frac{\partial \mathbf{g}_{\mathbf{R}}(\mathbf{y}_{\mathbf{N}}^{(\nu)})}{\partial \mathbf{y}_{\mathbf{N}}} \tag{8.2.6d}$$

Collecting (8.2.5) and (8.2.6), we find the linear Newton system as

$$\begin{bmatrix} \mathbf{L}_0^{(\nu)} & & & & \\ \mathbf{L}_1^{(\nu)} & \mathbf{R}_1^{(\nu)} & & & \\ & \ddots & \ddots & & \\ & & \mathbf{L}_N^{(\nu)} & \mathbf{R}_N^{(\nu)} & \\ & & & \mathbf{R}_{N+1}^{(\nu)} & \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{y}_0^{(\nu)} \\ \boldsymbol{\Delta}\mathbf{y}_1^{(\nu)} \\ \vdots \\ \boldsymbol{\Delta}\mathbf{y}_N^{(\nu)} \end{bmatrix} = - \begin{bmatrix} \mathbf{g}_L(\mathbf{y}_0^{(\nu)}) \\ \boldsymbol{\phi}_1(\mathbf{y}^{(\nu)}) \\ \vdots \\ \boldsymbol{\phi}_N(\mathbf{y}^{(\nu)}) \\ \mathbf{g}_R(\mathbf{y}_N^{(\nu)}) \end{bmatrix}. \tag{8.2.7}$$

We'll discuss an $O(N)$ algorithm for solving this block bidiagonal system in Section 8.5. For the moment, we'll consider an application.

Example 8.2.1 ([1], Section 5.1). Consider the second-order linear BVP

$$y'' - \lambda^2 y = \lambda^2 \cos^2 \pi x + 2\pi^2 \cos 2\pi x, \qquad 0 < x < 1, \qquad y(0) = y(1) = 0.$$

We'll write this as a first-order system by letting $y_1 = y$ and $y_2 = y/\lambda$ to get

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} 0 & \lambda \\ \lambda & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda \cos^2 \pi x + \frac{2\pi^2}{\lambda} \cos 2\pi x \end{bmatrix}.$$

The exact solution of this problem is

$$\mathbf{y}(x) = \begin{bmatrix} \frac{e^{-\lambda(1-x)} + e^{-\lambda x}}{1+e^{-\lambda}} - \cos^2 \pi x \\ \frac{e^{-\lambda(1-x)} - e^{-\lambda x}}{1+e^{-\lambda}} + \frac{\pi}{\lambda} \sin 2\pi x \end{bmatrix}.$$

This problem is similar to one that was addressed in Section 7.1. We recall that when $\lambda \gg 1$ there are boundary layers where the solution varies rapidly near both $x = 0$ and $x = 1$. The solution varies sinusoidally in the interior (Figure 8.2.2). Hence, this is a boundary value equivalent of a stiff problem.

For this linear problem

$$\mathbf{f}_y(x, \mathbf{y}) = \begin{bmatrix} 0 & \lambda \\ \lambda & 0 \end{bmatrix}.$$

The use of (8.2.6) reveals

$$\mathbf{L}_i^{(\nu)} = \mathbf{L}_i = -\frac{1}{h_i} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 0 & \lambda \\ \lambda & 0 \end{bmatrix} = \begin{bmatrix} -1/h_i & -\lambda/2 \\ -\lambda/2 & -1/h_i \end{bmatrix},$$

$$\mathbf{R}_i^{(\nu)} = \mathbf{R}_i = \begin{bmatrix} 1/h_i & -\lambda/2 \\ -\lambda/2 & 1/h_i \end{bmatrix}, \qquad i = 1, 2, \ldots, N.$$

Additionally,

$$\mathbf{L}_0^{(\nu)} = \mathbf{L}_0 = [1\ 0], \qquad \mathbf{R}_{N+1}^{(\nu)} = \mathbf{R}_{N+1} = [1\ 0].$$

Ascher *et al.* [1] solve a problem with $\lambda = 50$ by the midpoint and trapezoidal rules. The maximum error

$$\|e_{1,i}\|_\infty = \max_{0<i<N} |y_1(x_i) - y_{1,i}|$$

of the first component of the solution is used as an accuracy measure. Results using uniform meshes are shown in Table 8.2.1. Although the convergence rate appears to be

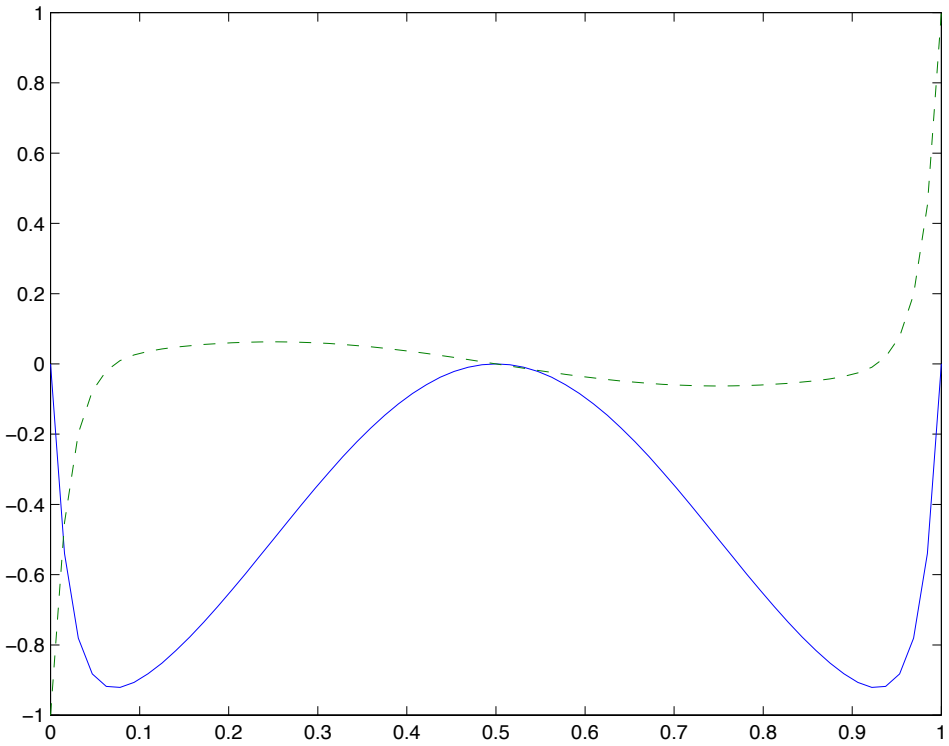

Figure 8.2.2: Exact solution of Example 8.2.1 with $\lambda = 50$. The solid line is y_1 and the dashed line is y_2.

quadratic, the accuracy is poor. With boundary layers near both ends (Figure 8.2.2), accuracy may be improved by using a nonuniform mesh that is concentrated near the endpoints. Ascher *et al.* [1] did this and their results are shown in Table 8.2.2. The results are somewhat better with convergence again appearing to be at a quadratic rate. Thus, a nonuniform mesh with the box scheme does not appear to reduce the convergence rate relative to that on a uniform mesh. This would not be the case with central differencing. Results computed with the trapezoidal rule are approximately three times more accurate than those computed by the midpoint rule on a nonuniform mesh.

8.3 Higher-Order Methods and Deferred Corrections

Let us once again consider the solution of the nonlinear BVP (8.2.1) by the trapezoidal rule (8.2.3). To begin, we'll recall that the local discretization error (Definition 6.3.1) of

N	Midpoint Rule	Trapezoidal Rule
10	0.47	0.44
20	0.20	0.19
40	0.057	0.056
80	0.01	0.012

Table 8.2.1: Errors in the solution of Example 8.2.1 using the midpoint and trapezoidal rules on uniform meshes.

N	Midpoint Rule	Trapezoidal Rule
10	0.16	0.55
20	0.042	0.015
40	0.0096	0.0036
80	0.0024	0.00091

Table 8.2.2: Errors in the solution of Example 8.2.1 using the midpoint and trapezoidal rules on nonuniform meshes.

the trapezoidal rule is

$$\boldsymbol{\tau}_i = \boldsymbol{\phi}_i(\mathbf{y}(x)) = \frac{\mathbf{y}(x_i) - \mathbf{y}(x_{i-1})}{h_i} - \frac{\mathbf{f}(x_i, \mathbf{y}(x_i)) + \mathbf{f}(x_{i-1}, \mathbf{y}(x_{i-1}))}{2}. \tag{8.3.1}$$

As usual, this may be put in a more explicit form by using Taylor's series expansions.

Lemma 8.3.1. *If* $\mathbf{f}(x, \mathbf{y}) \in C^{2K+2}(a, b)$ *then*

$$\boldsymbol{\tau}_i = \sum_{k=1}^{K} h_i^{2k} \mathbf{T}_k(\mathbf{y}(x_{i-1/2})) + O(h_i^{2K+2}) \tag{8.3.2a}$$

where

$$\mathbf{T}_k(\mathbf{y}(x)) = -\frac{k}{2^{2k-1}(2k+1)!} \mathbf{f}^{(2k)}(x, \mathbf{y}(x)). \tag{8.3.2b}$$

Proof. Expand the local discretization error in a Taylor's series about $x_{i-1/2}$. □

Thus, the local discretization error of the trapezoidal rule has a Taylor's series expansion in even powers of the mesh spacing. The leading term of this expansion is

$$\boldsymbol{\tau}_i = -\frac{h_i^2}{12} \mathbf{f}''(x_{i-1/2}, \mathbf{y}(x_{i-1/2})).$$

The result (8.3.2) does not depend on the mesh being uniform, but having some mesh regularity will provide benefits.

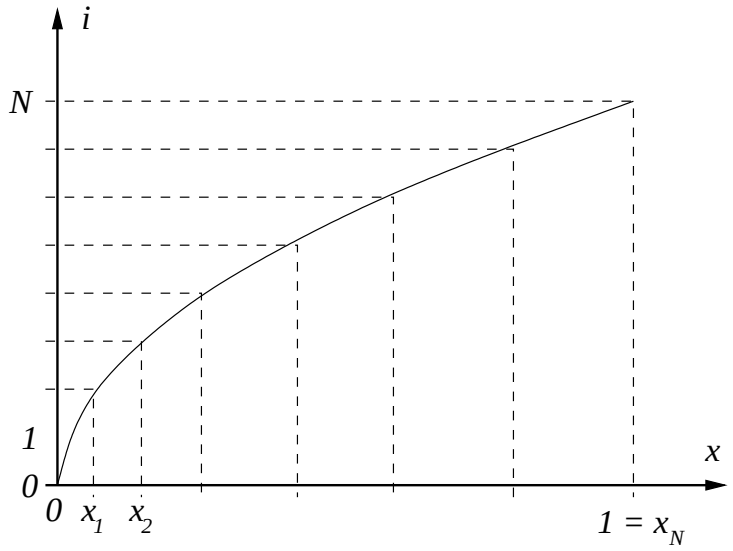

Figure 8.3.1: A mesh with quadratic grading.

Definition 8.3.1. *A sequence of meshes* $\{a = x_0 < x_1 < \ldots < x_N = b\}$ *with spacing* $h_i = x_i - x_{i-1}$, $i = 1, 2, \ldots, N$, *characterized by* $N \to \infty$ *is called quasi uniform if there exists a constant* $C > 0$ *independent of*

$$h = \frac{b-a}{N} \tag{8.3.3a}$$

such that

$$\frac{h}{h_i} \leq C, \qquad i = 1, 2, \ldots, N, \qquad N \to \infty. \tag{8.3.3b}$$

Example 8.3.1. Consider a quadratically graded mesh on $0 \leq x \leq 1$ (Figure 8.3.1)

$$x_i = (\frac{i}{N})^2, \qquad i = 0, 1, \ldots, N.$$

The mesh spacing for this example is

$$h_i = \frac{i^2 - (i-1)^2}{N^2} = \frac{2i-1}{N^2}.$$

Then,

$$\frac{h}{h_i} = \frac{1/N}{(2i-1)/N^2} = \frac{N}{2i-1}, \qquad i = 1, 2, \ldots, N.$$

The smallest subinterval is the first ($i = 1$); thus,

$$\frac{h}{h_i} \leq N, \qquad i = 1, 2, \ldots, N.$$

There is no bound as $N \to \infty$; therefore, this sequence of quadratically graded meshes is not quasi uniform.

The presence of a quasi-uniform mesh enables us to show that the global error has an expansion in even powers of the mesh spacing.

Theorem 8.3.1. *Suppose that there is a unique solution* $\mathbf{y}(x) \in C^{2K+2}(a,b)$ *of the linear BVP*

$$\mathbf{y}' = \mathbf{A}(x)\mathbf{y} + \mathbf{b}(x), \qquad a < x < b, \tag{8.3.4a}$$

$$\mathbf{L}\mathbf{y}(a) = \mathbf{l}, \qquad \mathbf{R}\mathbf{y}(b) = \mathbf{r}. \tag{8.3.4b}$$

Let finite difference solutions of (8.3.4) be generated by the trapezoidal rule on a sequence of quasi-uniform meshes $\{a = x_0 < x_1 < \ldots < x_N = b\}$, $N \to \infty$. *Then the global error satisfies*

$$\mathbf{y}(x_i) - \mathbf{y}_i = \sum_{k=1}^{K} \mathbf{d}_k(x_i) h_i^{2k} + O(h^{2K+2}), \qquad i = 0, 1, \ldots, N, \tag{8.3.5a}$$

where $\mathbf{d}_k(x)$, $k = 1, 2, \ldots, K$, *are smooth functions satisfying*

$$\mathbf{d}_k' - \mathbf{A}(x)\mathbf{d}_k = \mathbf{T}_k(\mathbf{y}(x)) - \sum_{l=1}^{k-1} \mathbf{T}_l(\mathbf{d}_k - l(x)), \tag{8.3.5b}$$

$$\mathbf{L}\mathbf{d}_k(a) = \mathbf{R}\mathbf{d}_k(b) = \mathbf{0}. \tag{8.3.5c}$$

Proof. ([1], Section 5.5). Given a trapezoidal rule solution of (8.3.4) on a mesh $\{x_0 < x_1 < \ldots < x_N\}$, let

$$\mathbf{v}_i = \mathbf{y}(x_i) - \mathbf{y}_i - \sum_{k=1}^{K} \mathbf{d}_k(x_i) h_i^{2k}, \qquad i = 0, 1, \ldots, N.$$

Our task is to show that $|\mathbf{v}|_\infty = O(h^{2K+2})$ where $\mathbf{v} = [\mathbf{v}_1,\ \mathbf{v}_2,\ \ldots,\ \mathbf{v}_N]^T$. Utilizing (8.1.2a), (8.1.3), and the linearity of the problem

$$\boldsymbol{\phi}_i(\mathbf{v}_i) = \boldsymbol{\phi}_i(\mathbf{y}(x_i) - \mathbf{y}_i) - \sum_{k=1}^{K} \boldsymbol{\phi}_i(\mathbf{d}_k(x_i)) h_i^{2k} = \boldsymbol{\tau}_i - \sum_{k=1}^{K} \boldsymbol{\phi}_i(\mathbf{d}_k(x_i)) h_i^{2k}.$$

Using the expansion (8.3.2a) for the local discretization error

$$\boldsymbol{\phi}_i(\mathbf{v}_i) = \sum_{k=1}^{K} h_i^{2k}[\mathbf{T}_k(\mathbf{y}(x_{i-1/2})) - \boldsymbol{\phi}_k(\mathbf{d}_k(x_i))] + O(h^{2K+2}). \tag{8.3.6}$$

Since $\mathbf{y}(x)$ is smooth, the data $\mathbf{A}(x)$ and $\mathbf{b}(x)$ must also be smooth. With (8.3.2b), this further implies that $\mathbf{T}_k$, $k = 1, 2, \ldots, K$, are smooth. These results can be used with (8.3.5b,c) to inductively show that $\mathbf{d}_k(x) \in C^{2(K-k)+3}$. With this level of smoothness, we can replace $\mathbf{y}(x)$ and K in (8.3.1) and (8.1.3) by $\mathbf{d}_k(x)$ and $K - k$ to obtain

$$\boldsymbol{\phi}_k(\mathbf{d}_k(x_i)) = \mathbf{d}_k'(x_i) - \mathbf{A}(x_i)\mathbf{d}_k(x_i) + \sum_{l=1}^{K-k} h_i^{2l}\mathbf{T}_l[\mathbf{d}_k(x_{i-1/2})] + O(h^{2(K-k+1)}).$$

Using (8.3.5b)

$$\boldsymbol{\phi}_k(\mathbf{d}_k(x_i)) = \mathbf{T}_k(\mathbf{y}(x_{i-1/2})) - \sum_{l=1}^{k-1} \mathbf{T}_l(\mathbf{d}_{k-l}(x_{i-1/2})) + \sum_{l=1}^{K-k} h_i^{2l}\mathbf{T}_l(\mathbf{d}_k(x_{i-1/2})) + O(h^{2(K-k+1)}).$$

Substituting this result into (8.3.6)

$$\boldsymbol{\phi}_i(\mathbf{v}_i) = \sum_{k=1}^{K} h_i^{2k}[\sum_{l=1}^{k-1} \mathbf{T}_l(\mathbf{d}_{k-l}(x_{i-1/2})) - \sum_{l=1}^{K-k} h_i^{2l}\mathbf{T}_l(\mathbf{d}_k(x_{i-1/2})) + O(h^{2(K-k+1)})].$$

All but the highest powers of h_i vanish leaving

$$\boldsymbol{\phi}_i(\mathbf{v}_i) = O(h^{2K+2}).$$

Ascher *et al.* [1] show that the operator $\boldsymbol{\phi}_i$ is stable. We'll omit their proof, but use the result with the homogeneous boundary data (8.3.5c) to infer that $\mathbf{v}_i = O(h^{2K+2})$, $i = 0, 1, \ldots, N$. □

Remark 1. The results appear to hold for nonlinear problems with smooth solutions and for graded meshes that are not quasi uniform.

Knowing that the global error of the trapezoidal rule has an expansion in even powers of the mesh spacing, we can construct higher-order approximations by Richardson's extrapolation. Thus, we calculate two solutions $\mathbf{y}_i^h$, $i = 0, 1, \ldots, N$, and $\mathbf{y}_{i/2}^{h/2}$, $i = 0, 1, \ldots, 2N$, with spacing h_i and $h_i/2$, $i = 0, 1, \ldots, N$, respectively. Using (8.3.5a) we have

$$\mathbf{y}(x_i) - \mathbf{y}_i^h = \sum_{k=1}^{K} \mathbf{d}_k(x_i)h_i^{2k} + O(h^{2K+2}), \qquad i = 0, 1, \ldots, N,$$

$$\mathbf{y}(x_i) - \mathbf{y}_i^{h/2} = \sum_{k=1}^{K} \mathbf{d}_k(x_i)(\frac{h_i}{2})^{2k} + O(h^{2K+2}), \qquad i = 0, 1, \ldots, N.$$

Subtracting the two results, we obtain an approximation of the error as

$$\mathbf{d}_1(x_i)h_i^2 = \frac{4}{3}(\mathbf{y}_i^{h/2} - \mathbf{y}_i^h) + O(h^2) \tag{8.3.7a}$$

The approximation can be added to, e.g., $\mathbf{y}_i^{h/2}$, $i = 0, 1, \ldots, N$, to obtain the higher-order solution

$$\hat{\mathbf{y}}_i^{h/2} = \mathbf{y}_i^{h/2} + \frac{\mathbf{y}_i^{h/2} - \mathbf{y}_i^h}{3} + O(h^4). \tag{8.3.7b}$$

This process can be repeated to eliminate successively higher-order terms is the error expansion (8.3.5a). Instead of doing this, however, we'll describe the alternate approach of deferred corrections. Consider a solution $\mathbf{y}_i$, $i = 0, 1, \ldots, N$, of (8.2.3). Using (8.3.1), we know that this solution satisfies

$$\boldsymbol{\phi}_i(\mathbf{y}(x_i)) = \boldsymbol{\tau}_i(\mathbf{y}). \tag{8.3.8a}$$

Suppose that $\hat{\boldsymbol{\tau}}_i(\mathbf{y}))$ is an $O(h^p)$ approximation of $\boldsymbol{\tau}_i$, then the solution of

$$\boldsymbol{\phi}_i(\hat{\mathbf{y}}_i) = \hat{\boldsymbol{\tau}}_i(\mathbf{y}), \qquad i = 1, 2, \ldots, N-1, \qquad \mathbf{g}_L(\hat{\mathbf{y}}_0) = \mathbf{g}_R(\hat{\mathbf{y}}_N) = 0, \tag{8.3.8b}$$

is an $O(h^p)$ approximation of $\mathbf{y}(x)$. This process can be repeated, as shown in Figure 8.3.2, with successively better approximations of the local discretization error. Some comments about the procedure follow.

1. Unlike Richardson's extrapolation, the same mesh is used for the entire sequence of computations.

2. $\hat{\boldsymbol{\tau}}_i^K$ is an $O(h^{2K+2})$ approximation of the local discretization error; hence, it is an $O(h^{2K+2})$ approximation of the first K terms in (8.3.2a).

3. Likewise, $\mathbf{y}_i^{(K)}$, $i = 0, 1, \ldots, N$, is an $O(h^{2K+2})$ approximation of $\mathbf{y}(x)$.

4. $\hat{\boldsymbol{\tau}}_i^{(K+1)}(\mathbf{y}^{(K)})$ is an *a posteriori* estimate of the local discretization error of the solution $\mathbf{y}_i^{(K)}$, $i = 0, 1, \ldots, N$.

procedure deferred_correction;
  begin
    $\hat{\boldsymbol{\tau}}_i^{(0)}(\mathbf{y}^{(-1)}) = 0, \qquad i = 1, 2, \ldots, N;$
    $K := 0;$
    while accuracy not sufficient **do**
      begin
      Solve $\boldsymbol{\phi}_i(\mathbf{y}_i^{(K)}) = \hat{\boldsymbol{\tau}}_i^{(K)}(\mathbf{y}^{(K-1)}), \qquad i = 1, 2, \ldots, N-1,$
      $\mathbf{g}_L(\mathbf{y}_0^{(K)}) = \mathbf{g}_R(\mathbf{y}_N^{(K)}) = 0;$
      Calculate $\hat{\boldsymbol{\tau}}_i^{K+1}(\mathbf{y}^{(K)});$
      $K := K + 1$
    end
  end { deferred_correction };

Figure 8.3.2: Algorithm for deferred corrections.

5. Lentini and Pereyra [7] implemented this procedure in a finite-difference code called *PASVA* for the solution of two-point boundary value problems.

It remains to compute the approximation $\hat{\boldsymbol{\tau}}_i^{(K)}(\mathbf{y}^{(K-1)})$, $i = 0, 1, \ldots, N$. This can be done by passing a $(2K+1)$ th-degree polynomial through the $2K$ points neighboring x_i and interpolating $(x_j, \mathbf{f}(x_j, \mathbf{y}_j^{(K-1)}))$ at these points. We obtain an approximation $\hat{\mathbf{T}}_k^K(\mathbf{y}^{(K-1)})$ of $\mathbf{T}_k(\mathbf{y}(x_{i-1/2}))$ by differentiating the interpolating polynomial. The result is

$$\hat{\boldsymbol{\tau}}_i^{(K)}(\mathbf{y}^{(K-1)}) = \sum_{k=1}^{K} h_i^{2k} \hat{\mathbf{T}}_k^{(K)}(\mathbf{y}^{(K-1)}(x_{i-1/2})).$$

The interpolation formulas can be complex and, typically, a mixture of centered, forward, and backward difference approximations are used.

Example 8.3.2. When $K = 1$ we require $\hat{\boldsymbol{\tau}}_i^{(1)}(\mathbf{y}^{(0)})$ to be an $O(h^4)$ accurate approximation of τ_i. This can be done with a cubic polynomial approximation of

$$\mathbf{T}_1(\mathbf{y}(x_{i-1/2})) = -\frac{1}{12}\mathbf{f}''(x_{i-1/2}, \mathbf{y}(x_{i-1/2})).$$

Let the cubic polynomial interpolate $(x_j, \mathbf{f}(x_j, \mathbf{y}_j^{(0)}))$, $j = i-2, i-1, i, i+1$. If the points are equally spaced then

$$\hat{\boldsymbol{\tau}}_i^{(1)}(\mathbf{y}^{(0)}) = h^2 \hat{\mathbf{T}}_i^{(1)} = \frac{1}{24}(-\mathbf{f}_{i-2} + \mathbf{f}_{i-1} + \mathbf{f}_i - \mathbf{f}_{i+1}).$$

This formula would have to be modified near the boundaries.

Problems

1. Consider a logarithmically graded mesh

$$x_i = \frac{\ln(1+\frac{i}{N})}{\ln 2}, \qquad i = 0, 1, \ldots, N.$$

   This grading is less severe than the quadratic grading of Example 8.3.1. Is this mesh quasi uniform?

8.4 Adaptive Mesh Selection

We seek to solve the nonlinear BVP (8.2.1) to a specified degree of accuracy in an adaptive manner by (i) computing a preliminary solution on a coarse mesh, (ii) estimating the discretization error of this solution, and (iii) recursively refining the mesh where the accuracy is not sufficient. Thus, in the end, we would only use a fine mesh where the solution varied rapidly. A simple example serves to illustrate the principle.

Example 8.4.1. Consider the linear two-point BVP

$$y'' + \lambda y' = 0, \qquad 0 < x < 1, \qquad y(0) = 1, \qquad y(1) = 1/2,$$

which has the exact solution

$$y(x) = \frac{1 + e^{-\lambda x} - 2e^{-\lambda}}{2(1 - e^{-\lambda})}.$$

Assuming that $0 < 1/\lambda \ll 1$, this is approximately

$$y(x) \approx \frac{1}{2}(1 + e^{-\lambda x})$$

As shown in Figure 8.4.1, the solution has a boundary layer near $x = 0$ and we might be able to improve both accuracy and efficiency by using a mesh that is concentrated in the boundary layer.

Once again, we'll focus on the trapezoidal rule solution of (8.2.1) which is

$$\boldsymbol{\phi}_i(\mathbf{y}) = \frac{\mathbf{y}_i - \mathbf{y}_{i-1}}{h_i} - \frac{\mathbf{f}(x_i, \mathbf{y}_i) + \mathbf{f}(x_{i-1}, \mathbf{y}_{i-1})}{2}, \qquad i = 1, 2, \ldots, N, \tag{8.4.1a}$$

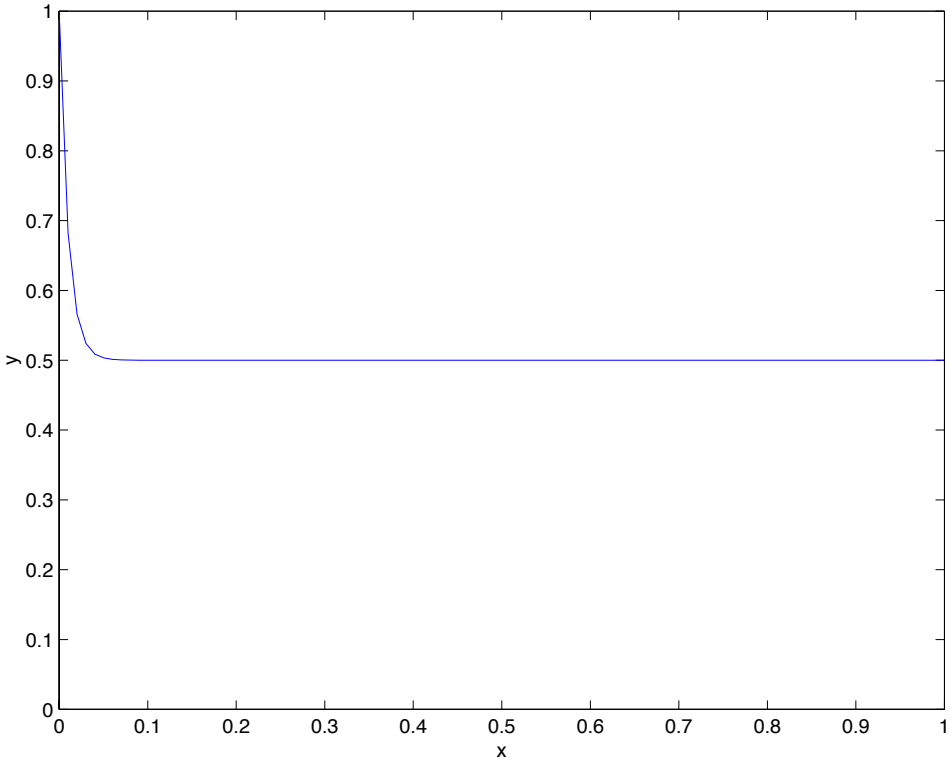

Figure 8.4.1: The solution of Example 8.4.1 with $\lambda = 100$.

$$\mathbf{g}_L(\mathbf{y}_0) = \mathbf{g}_R(\mathbf{y}_N) = 0. \tag{8.4.1b}$$

Using (8.3.2a)

$$\boldsymbol{\tau}_i = -\frac{h_i^2}{12}\mathbf{f}''(x_{i-1/2}, \mathbf{y}(x_{i-1/2})) + O(h^4). \tag{8.4.1c}$$

While concentrating on the trapezoidal rule, we'll consider a somewhat more abstract setting that will be useful when working with, e.g., high-order solutions *via* Richardson's extrapolation or deferred corrections. Thus, let e_i be an "error indicator" on the subinterval $[x_{i-1}, x_i)$. Some possibilities for e_i are:

1. An approximation of the local discretization error. For the trapezoidal rule this would be
$$e_i = \frac{h_i^2}{12}|\mathbf{f}''(x_{i-1/2}, \mathbf{y}_{i-1/2})|_\infty.$$

2. An approximation of the global discretization error

$$e_i = C_s h_i^s \|\mathbf{y}^{(s)}(\cdot)\|_{i,\infty}$$

where C_s is a known constant that depends upon the method and $\| \ \|_{i,\infty}$ is the maximum norm restricted to $[x_{i-1}, x_i)$.

3. An *ad hoc* metric. One could use a scaled approximation of the gradient or curvature, e.g.,

$$e_i = h_i \|\delta \mathbf{y}_{i-1/2}\|$$

where δ is the central difference operator (Table 6.3.3).

The key ingredient is that the error indicator be small where the solution is well-resolved and large where additional resolution is necessary.

Given a mesh of $N+1$ points, we try to determine the coordinates $\{x_0 < x_1 < \ldots < x_N\}$ such that some global metric of solution quality is minimized. We'll illustrate a solution of this problem by determining the mesh such that

$$|\mathbf{e}|_\infty = \max_{1 \le i \le N} |e_i|$$

is minimized. Although still in a rather abstract setting, de Boor [3], Chapter XIV, found a rather simple solution to this extremal problem. He assumed the error indicator to be a linear function of the mesh spacing, i.e.,

$$e_i = h_i w_i \tag{8.4.2}$$

where w_i is a weighting that is assumed to be independent of the mesh spacing. Weighting could be obtained by taking roots of other more traditional measures. For example, taking the $s\ th$ root of a global discretization error on $[x_{i-1}, x_i)$ measure might produce

$$e_i = h_i [C_s \|\mathbf{y}^{(s)}(\cdot)\|_{i,\infty}]^{1/s}.$$

The solution of the minimization problem is to determine the mesh such that e_i is the same on every subinterval $[x_{i-1}, x_i)$, $i = 1, 2, \ldots, N$. With this solution, the minimization problem is often called an "equidistribution" or "equilibration" problem. With

$$e_i = \lambda, \qquad i = 1, 2, \ldots, N, \tag{8.4.3a}$$

the equidistributing mesh is calculated from (8.4.2) as

$$h_i = \frac{\lambda}{w_i}, \qquad i = 1, 2, \ldots, N, \tag{8.4.3b}$$

where

$$h_i = x_i - x_{i-1}. \tag{8.4.3c}$$

It remains to calculate λ, and this can be done by summing (8.4.3b)

$$\sum_{i=1}^{N} h_i = b - a = \lambda \sum_{i=1}^{N} \frac{1}{w_i};$$

thus,

$$\lambda = \frac{b-a}{\sum_{i=1}^{N} 1/w_i}. \tag{8.4.3d}$$

Let us generalize the result and the procedure to continuous functions.

Definition 8.4.1. *A function $w(x, \mathbf{y}(x))$ is a monitor function if it is integrable and $w(x, \mathbf{y}(x)) \geq \epsilon$, $x \in (a, b)$, for some $\epsilon > 0$.*

Definition 8.4.2. *A mesh $\{a = x_0 < x_1 < \ldots < x_N = b\}$ is equidistributing with respect to a monitor function $w(x, \mathbf{y}(x))$ if there exists a constant $\lambda > 0$ such that*

$$\int_{x_{i-1}}^{x_i} w(x, \mathbf{y}(x))dx = \lambda, \qquad i = 1, 2, \ldots, N. \tag{8.4.4}$$

If the monitor function $w(x\mathbf{y}(x))$ is known, then it is as easy to find the equidistributing mesh in the continuous case as it was in the discrete case. Let us define

$$I(x) = \int_a^x w(\xi, \mathbf{y}(\xi))d\xi.$$

We observe that $I(x)$ is a monotonically increasing function of x since $w(x, \mathbf{y}(x)) > 0$. Now, the equidistribution problem (8.4.4) can be written as

$$I(x_i) = i\lambda, \qquad i = 1, 2, \ldots, N. \tag{8.4.5}$$

We first determine λ by setting $i = N$ in (8.4.5) to obtain

$$\lambda = \frac{I(x_N)}{N} = \int_a^b w(x, \mathbf{y}(x))dx. \tag{8.4.6}$$

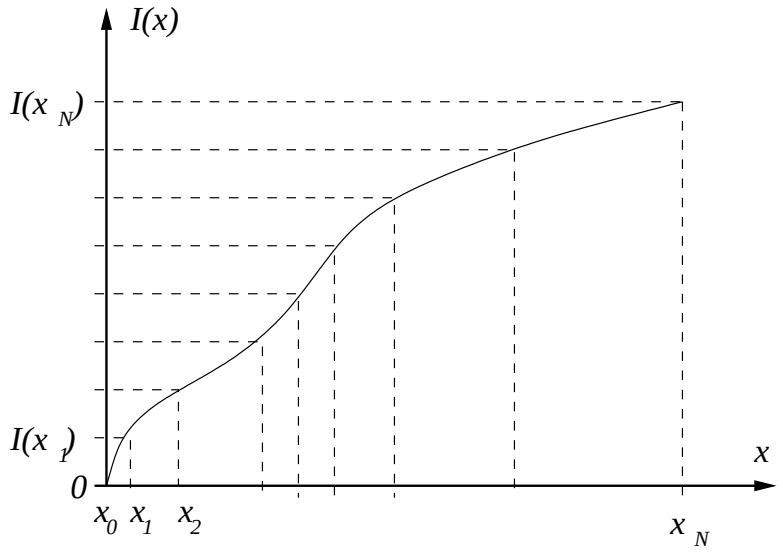

Figure 8.4.2: Determining an equidistributing mesh from a monitor function.

With λ known, the mesh is calculated by finding the roots of (8.4.5). The situation is shown in Figure 8.4.2. The roots of (8.4.5) can be determined sequentially without need of solving a vector system of nonlinear equations.

Unfortunately $w(x, \mathbf{y}(x))$ is typically a function of the solution of the BVP and, hence, it may only be known discretely with respect to a mesh. We'll view this mesh and solution as being preliminary and use them to calculate an equidistributing mesh that will subsequently be used to determine an improved solution. Thus, suppose that w_i, $i = 1, 2, \ldots, N$, has been determined by using an input mesh $\{x_0^0 < x_1^0 < \ldots < x_N^0\}$. Let us calculate a continuous approximation $w(x, \mathbf{y}(x))$ to w_i, $i = 1, 2, \ldots, N$, by polynomial interpolation. We need not do this particularly carefully since precise solutions of the equidistribution problem are not necessary. Assuming, for example, that $\|\mathbf{e}\|_\infty$ is a smooth function of x_i, $i = 0, 1, \ldots, N$, then, near a minimum, an $O(h)$ error in the placement of the mesh would only result in an $O(h^2)$ change in the value of $\|\mathbf{e}\|_\infty$.

Using these arguments, it would suffice to construct a piecewise constant approximation of w_i, $i = 1, 2, \ldots, N$, i.e.,

$$w(x, \mathbf{y}(x)) = w_i, \qquad x \in [x_{i-1}, x_i), \qquad i = 1, 2, \ldots, N.$$

This situation is shown in Figure 8.4.3. With a piecewise constant monitor function, $I(x)$

would be the piecewise linear function

$$I(x) = I(x_{j-1}^0) + w_j(x - x_{j-1}^0), \qquad x \in [x_{j-1}^0, x_j^0)$$

as shown in Figure 8.4.4. In this case, (8.4.5) can be solved exactly. For example, suppose

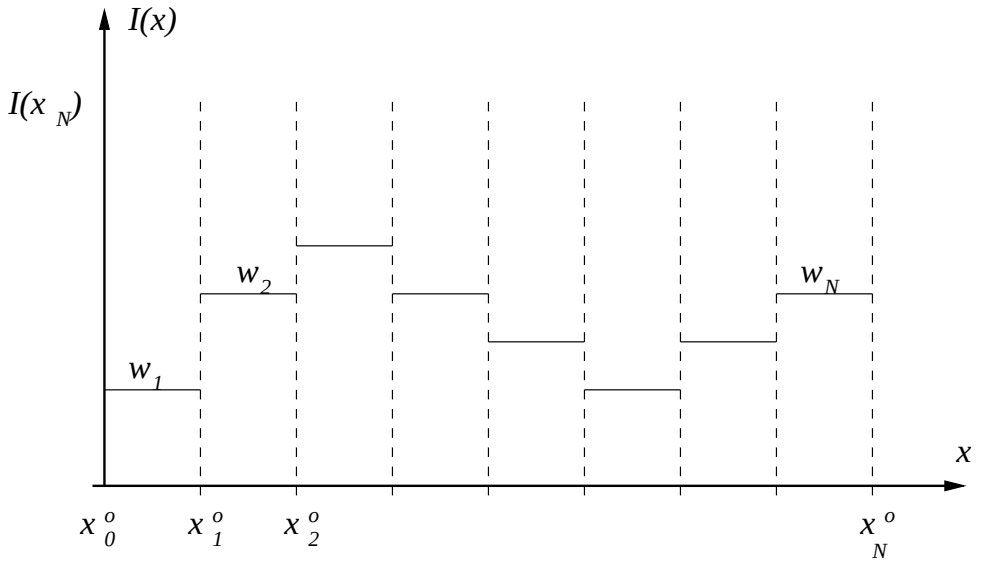

Figure 8.4.3: Piecewise constant monitor function.

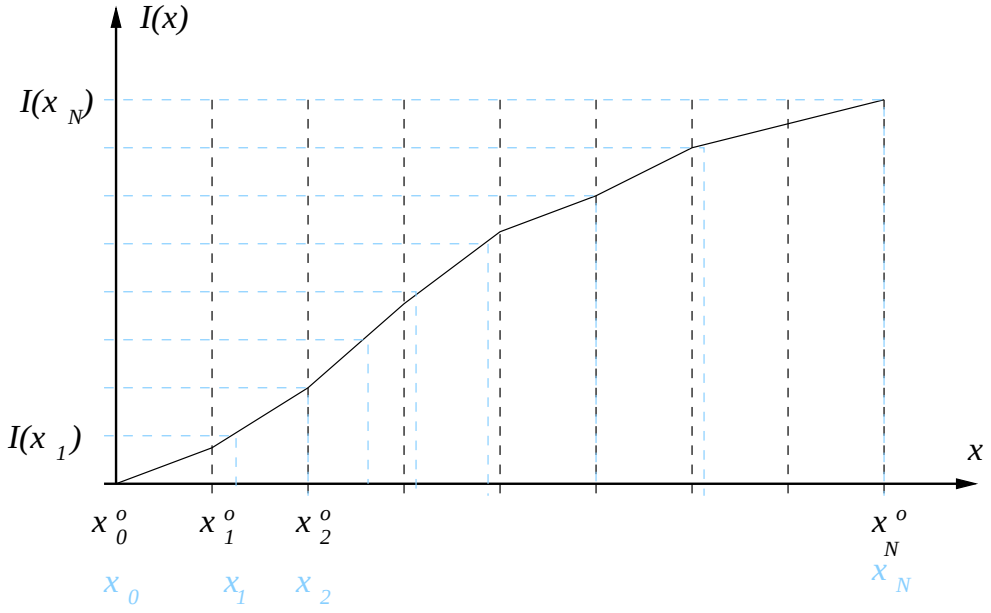

Figure 8.4.4: Piecewise linear function $I(x)$ based on a piecewise constant monitor function.

$$I(x_{j-1}^0) \le i\lambda < I(x_j),$$

then

$$x_i = x_{j-1}^0 + \frac{i\lambda - I(x_{j-1}^0)}{w_j}.$$

A piecewise linear interpolation of w_i, $i = 1, 2, \ldots, N$, is also possible, e.g.,

$$w(x, \mathbf{y}(x)) = w_{j-1}\frac{x_j^0 - x}{x_j^0 - x_{j-1}^0} + w_j\frac{x - x_{j-1}^0}{x_j^0 - x_{j-1}^0}.$$

In this case, $I(x)$ is a piecewise quadratic function of x and the roots of (8.4.5) can still be determined exactly [2].

Remark 1. You may think of using an iterative process where the equidistributing mesh obtained as described above is used as an input mesh to calculate a "better" equidistributing mesh, *etc.* Coyle *et al.* [2] show that this procedure is not likely to converge.

The dimensions of the input and output meshes need not be the same. One could, for example, calculate an estimate of the number of points needed in the output mesh to reduce the global error to a desired tolerance. Let us simplify our previous notation and suppose that an error indicator of the form

$$e_j = d_j(h_j^0)^s$$

is an estimate of the global error (in, e.g., the maximum norm) on the subinterval $[x_{j-1}^0, x_j^0)$. The constant d_j includes the dependence of the error on derivatives of the solution. If $e_i > TOL$ where TOL is the prescribed tolerance, we could calculate a new mesh spacing

$$h_j = \frac{h_j^0}{N_j}$$

and determine the refinement factor N_j such that

$$d_j h_j^s \approx TOL.$$

Thus,

$$N_j \approx h_j^0(\frac{d_j}{TOL})^{1/s}.$$

The N_j points could be added to the subinterval $[x_{j-1}^0, x_j^0)$ or used to determine the dimension of an equidistributing mesh. In a similar manner, points can be removed from subintervals that have small errors.

8.5 Solving Block Bidiagonal Linear Systems

In Section 8.2, we saw that the solution of a vector system of first-order differential equations by the trapezoidal rule required the solution of a block bidiagonal linear algebraic system of the form

$$\mathbf{A}\mathbf{y} = \mathbf{b} \tag{8.5.1a}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{L}_0 & & & & \\ \mathbf{L}_1 & \mathbf{R}_1 & & & \\ & \ddots & \ddots & & \\ & & \mathbf{L}_N & \mathbf{R}_N & \\ & & & \mathbf{R}_{N+1} & \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \\ \mathbf{b}_{N+1} \end{bmatrix}. \tag{8.5.1b}$$

The matrices $\mathbf{L}_i$ and $\mathbf{R}_i$, $i = 1, 2, \ldots, N$, have dimension $m \times m$, but the matrix $\mathbf{L}_0$ has dimension $l \times m$ and $\mathbf{R}_{N+1}$ has dimension $r \times m$ where $l + r = m$. Likewise, $\mathbf{b}_0$ is an l-vector, $\mathbf{b}_i$, $i = 1, 2, \ldots, N$, are m-vectors, and $\mathbf{b}_{N+1}$ is an r-vector. Each $\mathbf{y}_i$, $i = 0, 1, \ldots, N$, is an m-vector. The zero-nonzero structure of the matrix $\mathbf{A}$ is shown in Figure 8.5.1 for a case when $l = r = 1$ and $m = 2$.

Systems of this form also arise in conjunction with multiple shooting procedures (Section 7.2) and with collocation methods (Section9.1).

Keller [6] embedded the block bidiagonal matrix $\mathbf{A}$ in a block tridiagonal matrix as shown in Figure 8.5.2 and used the block tridiagonal algorithm [5] to solve the problem. The block tridiagonal algorithm is a matrix extension of the scalar tridiagonal algorithm that we considered in Section 6.3. Unfortunately, this procedure may not be stable without pivoting. While the algorithm can be implemented with pivoting, it induces some matrix fill-in, which is not desirable. We will discuss a stable procedure due to Varah [8] that involves alternate row and column elimination. For the 2×2 matrix shown in Figure 8.5.1, we take a suitable combination of the first two columns to create a zero in the $(1, 2)$ position of the matrix. The first two columns may be interchanged so that the largest element is the pivot. We then proceed to the second row and eliminate the $(3, 2)$ element by row elimination. Interchanges of the second and third rows ensure that the largest pivot element is chosen. The process continues in this manner and the

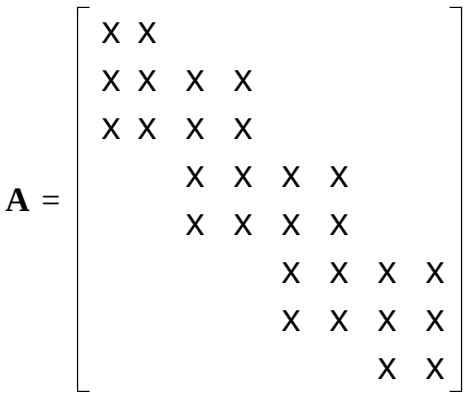

Figure 8.5.1: Structure of a 2×2 block bidiagonal matrix.

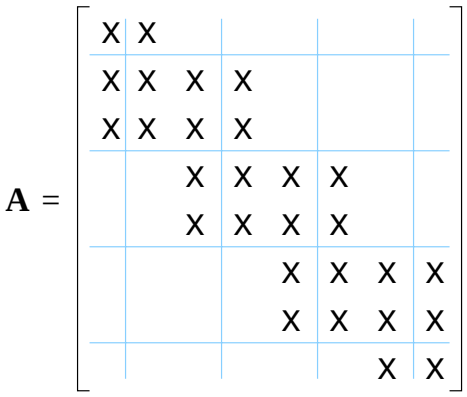

Figure 8.5.2: Embedding a 2×2 block bidiagonal matrix in a block tridiagonal matrix.

resulting system is shown in Figure 8.5.3. There is no fill-in and the algorithm is stable with the partial row and column pivoting discussed. Reduction of a general system is shown in Figure 8.5.4.

Recall, in Gaussian elimination row interchanges correspond to a reordering of the right hand side $\mathbf{b}$ while column interchanges correspond to a reordering of the unknowns $\mathbf{y}$. Row interchanges correspond to a factoring

$$\mathbf{PA} = \mathbf{LU} \tag{8.5.2a}$$

whereas column interchanges correspond to a factoring

$$\mathbf{AQ} = \mathbf{LU}. \tag{8.5.2b}$$

The matrices $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively, and $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices containing the row and column pivot information, respectively.

$$\mathbf{A} = \begin{bmatrix} \times & 0 & & & & & & \\ \times & \times & \times & \times & & & & \\ \times & 0 & \times & 0 & & & & \\ & & \times & \times & \times & \times & & \\ & & \times & 0 & \times & 0 & & \\ & & & & \times & \times & \times & \times \\ & & & & \times & 0 & \times & 0 \\ & & & & & & \times & \times \end{bmatrix}$$

Figure 8.5.3: Alternate row and column elimination of a 2×2 block bidiagonal matrix.

$$\mathbf{B} = \begin{bmatrix} \mathbf{L}^{(0)} & \mathbf{0} & & & & & & \\ \mathbf{X} & \mathbf{U}^{(0)} & \mathbf{X} & \mathbf{X} & & & & \\ \mathbf{X} & \mathbf{0} & \mathbf{L}^{(1)} & \mathbf{0} & & & & \\ & & \mathbf{X} & \mathbf{U}^{(1)} & \mathbf{X} & \mathbf{X} & & \\ & & \mathbf{X} & \mathbf{0} & \mathbf{L}^{(2)} & \mathbf{0} & & \\ & & & & \mathbf{X} & \mathbf{U}^{(2)} & \mathbf{X} & \mathbf{X} \\ & & & & \mathbf{X} & \mathbf{0} & \mathbf{L}^{(3)} & \mathbf{0} \\ & & & & & & \mathbf{X} & \mathbf{U}^{(3)} \end{bmatrix}$$

Figure 8.5.4: Alternate row and column elimination of block bidiagonal matrix with $N = 3$

The entire factorization has the form

$$\mathbf{PAQ} = \mathbf{LBU} \tag{8.5.3}$$

where the structure of $\mathbf{B}$ is shown in Figure 8.5.4. If $\mathbf{P}^{(i)}$, $\mathbf{Q}^{(i)}$, *etc.*, are the matrices introduced at the i *th* stage of the process, then

$$\mathbf{P} = \mathbf{P}^{(N)}\mathbf{P}^{(N-1)} \ldots \mathbf{P}^{(0)}, \tag{8.5.4a}$$

$$\mathbf{Q} = \mathbf{Q}^{(0)}\mathbf{Q}^{(1)} \ldots \mathbf{Q}^{(N)}, \tag{8.5.4b}$$

$$\mathbf{L} = \tilde{\mathbf{L}}^{(0)}\tilde{\mathbf{L}}^{(1)} \ldots \tilde{\mathbf{L}}^{(N)}, \tag{8.5.4c}$$

$$\mathbf{U} = \tilde{\mathbf{U}}^{(N)}\tilde{\mathbf{U}}^{(N-1)} \ldots \tilde{\mathbf{U}}^{(0)}. \tag{8.5.4d}$$

The matrices $\tilde{\mathbf{L}}^{(i)}$ and $\tilde{\mathbf{U}}^{(i)}$ have the same structure as $\mathbf{L}^{(i)}$ and $\mathbf{U}^{(i)}$, $i = 0, 1, \ldots, N$, but are expanded to the dimension $m(N+1)$ of $\mathbf{A}$.

Once the factorization is complete, the system (8.5.1) is solved by successively solving

$$\mathbf{Lt} = \mathbf{Pb}, \tag{8.5.5a}$$

$$\mathbf{Bz} = \mathbf{t}, \tag{8.5.5b}$$

$$\mathbf{Us} = \mathbf{z}, \tag{8.5.5c}$$

$$\mathbf{y} = \mathbf{Qs}. \tag{8.5.5d}$$

The systems (8.5.5a) and (8.5.5c) are solved by forward and backward substitution, respectively. The system (8.5.5d) is just a reordering of $\mathbf{s}$. Upon examination of Figure 8.5.4, we see that we may determine the unknowns in $\mathbf{z}$ corresponding to "odd" numbered blocks in $\mathbf{B}$ by forward substitution using the lower triangular matrices $\mathbf{L}^{(i)}$, $i = 0, 1, \ldots, N$. Knowing these unknowns, the remaining components of $\mathbf{z}$ may be determined by backward substitution using the "even" numbered blocks and the upper triangular matrices $\mathbf{U}^{(i)}$, $i = N, N-1, \ldots, 0$. Slightly different procedures are given in Ascher *et al.* [1], Section 7.2 and Diaz *et al.* [4].

Bibliography

[1] U.M. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM, Philadelphia, second edition, 1995.

[2] J.M. Coyle, J.E. Flaherty, and R. Ludwig. On the stability of mesh equidistribution strategies for time-dependent partial differential equations. *Journal of Computational Physics*, 62:26–39, 1986.

[3] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[4] J.C. Diaz, G. Fairweather, and P. Keast. Fortran packages for solving almost block diagonal linear systems by modified alternate row and column elimination. *ACM Transactions on Mathematical Software*, 9:358–375, 1981.

[5] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. John Wiley and Sons, New York, 1966.

[6] H.B. Keller. Accurate difference methods for linear ordinary differential systems subject to linear constraints. *SIAM J. Numer. Anal.*, 6:8–30, 1969.

[7] M. Lentini and V. Pereyra. An adaptive finite difference solver for nonlinear two point problems with mild boundary layers. *SIAM J. Numer. Anal.*, 14:91–111, 1977.

[8] J.M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.*, 13:71–75, 1976.

Chapter 9

Collocation Methods

9.1 Introduction

Let's continue the discussion of collocation methods with the second-order linear BVP

$$\mathcal{L}y = y'' + p(x)y' + q(x)y = r(x), \qquad a < x < b, \tag{9.1.1a}$$

$$y(a) = A, \qquad y(b) = B. \tag{9.1.1b}$$

To pick up where we left off in Section 6.4, consider an approximate solution

$$Y(x) = \sum_{i=0}^{N} [c_i \phi_i^3(x) + d_i \omega_i^3(x)] \tag{9.1.2a}$$

involving the cubic Hermite basis

$$\phi_i^3(x) = \begin{cases} 1 - 3(\frac{x-x_i}{h_i})^2 - 2(\frac{x-x_i}{h_i})^3, & \text{if } x_{i-1} \le x < x_i \\ 1 - 3(\frac{x-x_i}{h_{i+1}})^2 + 2(\frac{x-x_i}{h_{i+1}})^3, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases}, \tag{9.1.2b}$$

$$\omega_i^3(x) = \begin{cases} (x - x_i)[1 + \frac{x-x_i}{h_i}]^2, & \text{if } x_{i-1} \le x < x_i \\ (x - x_i)[1 - \frac{x-x_i}{h_{i+1}}]^2, & \text{if } x_i \le x < x_{i+1} \\ 0, & \text{otherwise} \end{cases}. \tag{9.1.2c}$$

Recall, that $[a, b]$ is divided into N subintervals with each having contributions from four non-trivial basis functions (Figure 9.1.1). In particular, the restriction of the basis to $[x_{i-1}, x_i]$ involves non-trivial contributions from $\phi_{i-1}^3(x)$, $\omega_{i-1}^3(x)$, $\phi_i^3(x)$, and $\omega_i^3(x)$

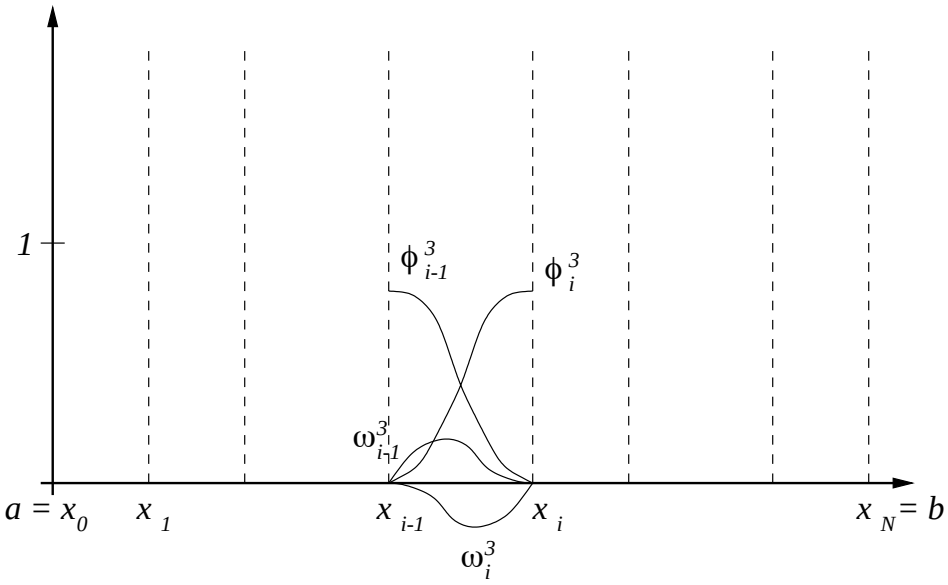

Figure 9.1.1: Geometry for the collocation solution of (9.1.1) showing the restriction of the cubic Hermite polynomial basis to the subinterval $[x_{i-1}, x_i]$.

In order to obtain approximate solutions of (9.1.1) using (9.1.2), we collocate at two points ξ_{i1}, ξ_{i2} per subinterval and satisfy the boundary conditions. Thus, the unknown coefficients c_i, d_i, $i = 0, 1, ..., N$, are determined as the solution of

$$\mathcal{L}Y(\xi_{ij}) = r(\xi_{ij}), \qquad j = 1, 2, \qquad i = 1, 2, \ldots, N, \tag{9.1.3a}$$

$$Y(a) = A, \qquad Y(b) = B. \tag{9.1.3b}$$

The restriction of (9.1.2a) to the subinterval $[x_{i-1}, x_i)$ is

$$Y(x) = c_{i-1}\phi^3_{i-1}(x) + d_{i-1}\omega^3_{i-1}(x) + c_i\phi^3_i(x) + d_i\omega^3_i(x). \tag{9.1.4}$$

Substituting (9.1.4) into (9.1.3a)

$$\begin{bmatrix} \mathcal{L}Y(\xi_{i,1}) \\ \mathcal{L}Y(\xi_{i,2}) \end{bmatrix} = \mathbf{L}_i \begin{bmatrix} c_{i-1} \\ d_{i-1} \end{bmatrix} + \mathbf{R}_i \begin{bmatrix} c_i \\ d_i \end{bmatrix} = \begin{bmatrix} r(\xi_{i,1}) \\ r(\xi_{i,2}) \end{bmatrix}, \qquad i = 1, 2, \ldots, N, \tag{9.1.5a}$$

where

$$\mathbf{L}_i = \begin{bmatrix} \mathcal{L}\phi^3_{i-1}(\xi_{i,1}) & \mathcal{L}\omega^3_{i-1}(\xi_{i,1}) \\ \mathcal{L}\phi^3_{i-1}(\xi_{i,2}) & \mathcal{L}\omega^3_{i-1}(\xi_{i,2}) \end{bmatrix}, \qquad \mathbf{R}_i = \begin{bmatrix} \mathcal{L}\phi^3_i(\xi_{i,1}) & \mathcal{L}\omega^3_i(\xi_{i,1}) \\ \mathcal{L}\phi^3_i(\xi_{i,2}) & \mathcal{L}\omega^3_i(\xi_{i,2}) \end{bmatrix}. \tag{9.1.5b}$$

With (9.1.4), the boundary conditions (9.1.3b) become

$$\mathbf{L}_0 \begin{bmatrix} c_0 \\ d_0 \end{bmatrix} = A, \qquad \mathbf{R}_{N+1} \begin{bmatrix} c_N \\ d_N \end{bmatrix} = B, \tag{9.1.6a}$$

where

$$\mathbf{L}_0 = \mathbf{R}_{N+1} = [1\ 0]. \tag{9.1.6b}$$

Combining (9.1.5) and (9.1.6), we find

$$\begin{bmatrix} \mathbf{L}_0 & & & & \\ \mathbf{L}_1 & \mathbf{R}_1 & & & \\ & \ddots & \ddots & & \\ & & \mathbf{L}_N & \mathbf{R}_N & \\ & & & \mathbf{R}_{N+1} \end{bmatrix} \begin{bmatrix} c_0 \\ d_0 \\ c_1 \\ d_1 \\ \vdots \\ c_N \\ d_N \end{bmatrix} = \begin{bmatrix} A \\ r(\xi_{1,1}) \\ r(\xi_{1,2}) \\ r(\xi_{2,1}) \\ r(\xi_{2,2}) \\ \vdots \\ r(\xi_{N,1}) \\ r(\xi_{N,2}) \\ B \end{bmatrix}. \tag{9.1.7}$$

Thus, the $2(N+1)$ coefficients c_i, d_i, $i = 0, 1, \ldots, N$, are determined as the solution of a block bidiagonal matrix of dimension $2N$ with 2×2 blocks. This system may be solved by the methods of Section 8.5.

We can now ask if there is an optimal placement of the two collocation points on each subinterval that, e.g., minimizes the discretization error $y(x) - Y(x)$ in some norm. This question was answered in a landmark paper by de Boor and Swartz [3] and we will follow their analysis.

Consider the inner product

$$(v, u) = \int_a^b v(x)u(x)dx. \tag{9.1.8}$$

Let us assume that $u(x)$ and $v(x)$ are smooth except, perhaps for jump discontinuities at z_j, $j = 1, 2, \ldots, M-1$. Also let $z_0 = a$ and $z_M = b$. Consider

$$(v, \mathcal{L}u) = \int_a^b v[u'' + pu' + qu]dx \tag{9.1.9}$$

where the integral is interpreted as a sum of integrals over the subintervals (z_0, z_1), (z_1, z_2), $\ldots$, (z_{M-1}, z_M). For simplicity, we'll also assume that $A = B = 0$ and that $u(x)$ and $v(x)$ satisfy these conditions. Using (9.1.1), we integrate (9.1.9) by parts to obtain

$$(v, \mathcal{L}U) = \int_a^b [-v'u' - (pv)'u + qvu]dx + \sum_{j=1}^{M} \{v(x)u'(x) + p(x)v(x)u(x)\}_{z_{j-1}}^{z_j}.$$

Integrating the first term in the integrand by parts once more

$$(v, \mathcal{L}U) = \int_a^b [v'' - (pv)' + qv]u dx + \sum_{j=1}^{M} \{v(x)u'(x) - v'(x)u(x) + p(x)v(x)u(x)\}_{z_{j-1}}^{z_j}.$$

We can write this result in a simpler form by using the inner product notation (9.1.8) and by defining the jump in a function $q(x)$ at a point z as

$$[q(x)]_{x=z} = \lim_{\epsilon \to 0} q(z+\epsilon) - q(z-\epsilon). \tag{9.1.10}$$

With this notation, we have

$$(v, \mathcal{L}u) = (\mathcal{L}^T v, u) - \sum_{j=1}^{M-1} [vu' - v'u + pvu]_{z_j} \tag{9.1.11a}$$

where $\mathcal{L}^T$ is called the *adjoint operator* and satisfies

$$\mathcal{L}^T v = v'' - (pv)' + qv. \tag{9.1.11b}$$

Let us simplify matters somewhat by assuming that $u \in C^1(a,b)$ and $v \in C^0(a,b)$. Then, (9.1.11a) becomes

$$\sum_{j=1}^{M-1} [v'u]_{z_j} = (v, \mathcal{L}u) - (\mathcal{L}^T v, u). \tag{9.1.11c}$$

Definition 9.1.1. *The Green's function $G(\xi, x)$ for the operator $\mathcal{L}$ of (9.1.1) satisfies*

$$G(\xi, x) \in C^0(a,b) \times (a,b), \tag{9.1.12a}$$

$$\mathcal{L}^T G(\xi, x) = 0, \qquad (a, \xi^-) \cup (\xi^+, b), \qquad \xi^{\pm} = \lim_{\epsilon \to 0} \xi \pm \epsilon, \tag{9.1.12b}$$

$$G(\xi, a) = G(\xi, b) = 0, \tag{9.1.12c}$$

$$G_x(\xi, \xi^+) - G_x(\xi, \xi^-) = 1. \tag{9.1.12d}$$

When viewed as a function of x, the Green's function has a unit jump in its first derivative at the point ξ.

Now, if we choose $v(x)$ in (9.1.11c) as the Green's function $G(\xi, x)$ so that the only $(M = 2)$ discontinuity occurs at $z_1 = \xi$, we have

$$u(\xi) = (G(\xi, \cdot), \mathcal{L}u). \tag{9.1.13a}$$

If $u(x)$ is chosen as $y(x)$, the solution of (9.1.1), then

$$y(\xi) = (G(\xi, \cdot), \mathcal{L}y) = (G(\xi, \cdot), r). \tag{9.1.13b}$$

The relation (9.1.13a) holds for any smooth function $u(x)$ and not just the solution of (9.1.1). Since, for example, the collocation solution $Y(x) \in C^1(a, b)$, we can replace u in (9.1.13a) by Y to obtain

$$Y(\xi) = (G(\xi, \cdot), \mathcal{L}Y). \tag{9.1.13c}$$

Finally, letting

$$e(x) = y(x) - Y(x) \tag{9.1.14a}$$

denote the discretization error of the collocation solution, we subtract (9.1.13c) from (9.1.13b) to obtain

$$e(\xi) = (G(\xi, \cdot), \mathcal{L}e) = \int_a^b G(\xi, x)\mathcal{L}e(x)dx. \tag{9.1.14b}$$

Remark 1. Each result (9.1.13b), (9.1.13c), or (9.1.14b) relates a global quantity (y, Y, e) to its local counterpart $(\mathcal{L}y, \mathcal{L}Y, \mathcal{L}e)$ through the Green's function.

Let us write (9.1.14b) in the more explicit form

$$e(\xi) = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} G(\xi, x)\mathcal{L}e(x)dx = \sum_{i=1}^{N} e_i(\xi). \tag{9.1.15}$$

Suppose that $\xi \notin (x_{i-1}, x_i)$ so that $G(\xi, x)$ is smooth for $x \in (x_{i-1}, x_i)$. Write

$$\mathcal{L}e = \mathcal{L}(y - Y) = \mathcal{L}y - Pr + Pr - \mathcal{L}Y \tag{9.1.16a}$$

where Pr is a linear polynomial for $x \in (x_{i-1}, x_i)$ that interpolates both $\mathcal{L}y$ and $\mathcal{L}Y$ at the two collocation points $\xi_{i,1}$ and $\xi_{i,2}$ on this subinterval. Thus,

$$Pr = r(\xi_{i,1})\frac{x - \xi_{i,2}}{\xi_{i,1} - \xi_{i,2}} + r(\xi_{i,2})\frac{x - \xi_{i,1}}{\xi_{i,2} - \xi_{i,1}}. \tag{9.1.16b}$$

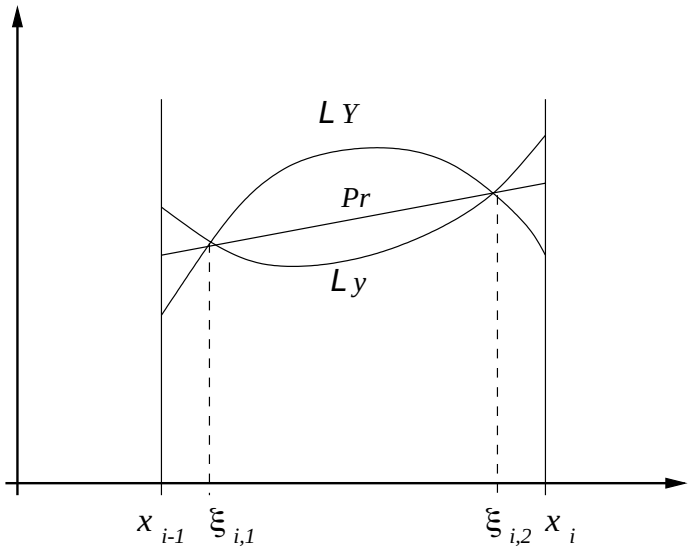

Figure 9.1.2: Functions $\mathcal{L}y$, $\mathcal{L}Y$, and Pr on a subinterval (x_{i-1}, x_i) not containing the point $x = \xi$.

The functions $\mathcal{L}y$, $\mathcal{L}Y$, and Pr are illustrated in Figure 9.1.2. The differences $\mathcal{L}y - Pr$ and $\mathcal{L}Y - Pr$ can be estimated using formulas for the error in linear interpolation [2] as

$$\mathcal{L}y - Pr = \frac{1}{2}(x - \xi_{i,1})(x - \xi_{i,2})(\mathcal{L}y)''(\eta_i) \tag{9.1.17a}$$

$$\mathcal{L}Y - Pr = \frac{1}{2}(x - \xi_{i,1})(x - \xi_{i,2})(\mathcal{L}Y)''(\zeta_i) \tag{9.1.17b}$$

where $\eta_i, \zeta_i \in (x_{i-1}, x_i)$.

Using (9.1.17) in (9.1.16) yields

$$e_i(\xi) = \int_{x_{i-1}}^{x_i} (x - \xi_{i,1})(x - \xi_{i,2}) g(\xi, x) dx, \qquad \xi \notin (x_{i-1}, x_i), \tag{9.1.18a}$$

where

$$g(\xi, x) = \frac{1}{2} G(\xi, x)[(\mathcal{L}y)''(\eta_i) - (\mathcal{L}Y)''(\zeta_i)]. \tag{9.1.18b}$$

where e_i was defined in (9.1.15). We bound (9.1.18a) as

$$|e_i(\xi)| \leq \int_{x_{i-1}}^{x_i} |x - \xi_{i,1}||x - \xi_{i,2}||g(\xi, x)| dx.$$

Since $|x - \xi_{i,j}| \leq h_i$, $j = 1, 2$, we have

$$|e_i(\xi)| \leq h_i^3 ||g(\xi, \cdot)||_{i,\infty}, \qquad \xi \notin (x_{i-1}, x_i), \tag{9.1.19a}$$

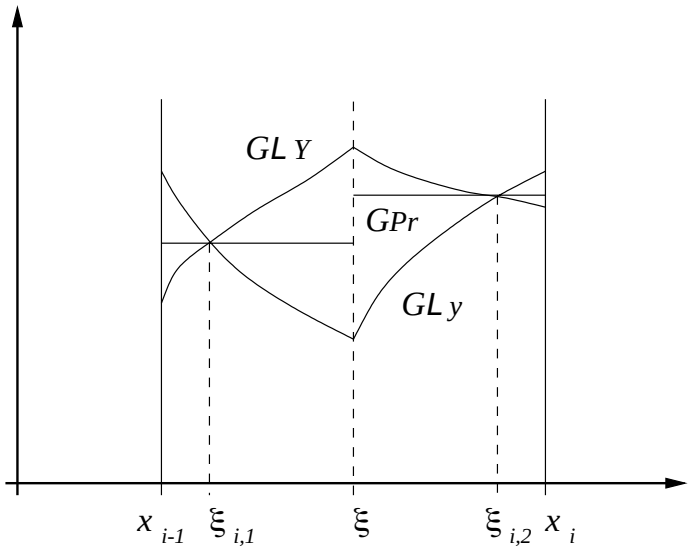

Figure 9.1.3: Functions $G\mathcal{L}y$, $G\mathcal{L}Y$, and GPr on a subinterval (x_{i-1}, x_i) containing the point $x = \xi$.

where

$$||f(\cdot)||_{i,\infty} = \max_{x_{i-1} \le x \le x_i} |f(x)|. \tag{9.1.19b}$$

Typically, one subinterval contains the point ξ where G_x is discontinuous. The analysis leading to (9.1.19) cannot be used on this subinterval since $G(\xi, x) \notin C^1(x_{i-1}, x_i)$. There are several ways of showing that $|e_i(\xi)|$ increases from $O(h_i^3)$ to $O(h_i^2)$ on this subinterval. We'll choose one which restricts ξ to lie between $\xi_{i,1}$ and $\xi_{i,2}$. In this case, we interpolate $G\mathcal{L}y$ and $G\mathcal{L}Y$ by piecewise constant functions

$$G(\xi, x)Pr = \begin{cases} G(\xi, \xi_{i,1})r(\xi_{i,1}), & \text{if } x_{i-1} \le x < \xi \\ G(\xi, \xi_{i,2})r(\xi_{i,2}), & \text{if } \xi \le x < x_i \end{cases} \tag{9.1.20}$$

as shown in Figure 9.1.3. The error in piecewise constant interpolation is

$$G\mathcal{L}y - GPr = \begin{cases} (x - \xi_{i,1})(G\mathcal{L}y)'(\eta_{i,1}), & \text{if } x_{i-1} \le x < \xi \\ (x - \xi_{i,2})(G\mathcal{L}y)'(\eta_{i,2}), & \text{if } \xi \le x < x_i \end{cases}.$$

A similar expression applies for $G\mathcal{L}Y - GPr$. Combining these results in the manner used to obtain (9.1.18a) yields

$$e_i(\xi) = \int_{x_{i-1}}^{\xi} (x - \xi_{i,1})\hat{g}(\xi, x)dx + \int_{\xi}^{x_i} (x - \xi_{i,2})\hat{g}(\xi, x)dx, \qquad \xi \in (x_{i-1}, x_i), \tag{9.1.21a}$$

where

$$\hat{g}(\xi,x)=\begin{cases}(G\mathcal{L}y)'(\eta_{i,1})-(G\mathcal{L}Y)'(\zeta_{i,1}), & \text{if } x_{i-1}\le x<\xi\\ (G\mathcal{L}y)'(\eta_{i,2})-(G\mathcal{L}Y)'(\zeta_{i,2}), & \text{if } \xi\le x<x_i\end{cases}. \tag{9.1.21b}$$

We bound (9.1.21a) as

$$|e_i(\xi)|\le\int_{x_{i-1}}^{\xi}|x-\xi_{i,1}||\hat{g}(\xi,x)|dx+\int_{\xi}^{x_i}|x-\xi_{i,2}||\hat{g}(\xi,x)|dx$$

or

$$|e_i(\xi)|\le h_i^2\|\hat{g}(\xi,\cdot)\|_{i,\infty}\qquad \xi\in(x_{i-1},x_i). \tag{9.1.21c}$$

We'll use the symbol $\|\hat{g}(\xi,\cdot)\|_{i,\infty}$ with the understanding that the maximum is computed on $(x_{i-1},\xi)\cup(\xi,x_i)$.

Finally, substituting (9.1.19a) and (9.1.21c) into (9.1.15) yields

$$|e(\xi)|\le h_j^2||\hat{g}(\xi,\cdot)||_{j,\infty}+\sum_{i=1,i\ne j}^{N}h_i^3\|g(\xi,\cdot)\|_{i,\infty},\qquad \xi\in(x_{j-1},x_j),$$

or

$$|e(\xi)|\le[h_j^2+\sum_{i=1,i\ne j}^{N}h_i^3]\|\tilde{g}(\xi,\cdot)\|_{\infty},\qquad \xi\in(x_{j-1},x_j),$$

where

$$\|\tilde{g}(\xi,\cdot)\|_{\infty}=\max(\|\hat{g}(\xi,\cdot)\|_{\infty},\|\hat{g}(\xi,\cdot)\|_{\infty}),\qquad \|f(\cdot)\|_{\infty}=\max_{1\le i\le N}\|f(\cdot)\|_{i,\infty}.$$

Letting

$$h=\max_{1\le i\le N}|h_i| \tag{9.1.22}$$

and observing that $Nh\le(b-a)$, we have

$$|e(\xi)|\le[h^2+(N-1)h^3]\|\tilde{g}(\xi,\cdot)\|_{\infty}\le h^2[1+(b-a)]\|\tilde{g}(\xi,\cdot)\|_{\infty}],\qquad \xi\in(x_{j-1},x_j),$$

or

$$|e(\xi)|\le Ch^2,\qquad \xi\in(x_{j-1},x_j), \tag{9.1.23}$$

where

$$C=(1+b-a)\|\tilde{g}(\xi,\cdot)\|_{\infty}.$$

If $\xi = x_j$, $j = 0, 1, \ldots, N$, then there is no discontinuity in the Green's function on any subinterval and the error is obtained from (9.1.19a) and (9.1.15) as

$$|e(x_j)| \leq \sum_{i=1}^{N} h_i^3 \|g(x_j, \cdot)\|_{i,\infty}, \qquad j = 0, 1, \ldots, N.$$

Following the steps leading to (9.1.23), we again find that

$$|e(x_j)| \leq Ch^2, \qquad j = 0, 1, \ldots, N. \tag{9.1.24}$$

Thus, the global and pointwise errors are both $O(h^2)$. This occurs because of the low polynomial degree and the arbitrary choice of the collocation points. With either higher-degree polynomials or a special choice of collocation points we can reduce the pointwise error relative to the global error. This phenomenon is called *nodal superconvergence.*

Definition 9.1.2. *Nodal superconvergence implies that the collocation solution on the mesh* $\{a = x_0 < x_1 < \cdots < x_N = b\}$ *converges to a higher order than it does globally.*

Now, let us resume the search for special collocation points. Consider the case when $\xi = x_j$, $j = 0, 1, \ldots, N$, so that $g(x_j, x)$ is smooth and expand it in a Taylor's series on the subinterval (x_{i-1}, x_i) to obtain

$$g(x_j, x) = g(x_j, x_{i-1/2}) + g_x(x_j, x_{i-1/2})(x - x_{i-1/2}) \quad + \quad \frac{1}{2} g_{xx}(x_j, \theta_i)(x - x_{i-1/2})^2,$$
$$\theta_i \in (x_{i-1}, x_i).$$

Substitute this expansion into (9.1.18a) to obtain

$$e_i(x_j) = \int_{x_{i-1}}^{x_i} (x - \xi_{i,1})(x - \xi_{i,2})[g(x_j, x_{i-1/2}) + g_x(x_j, x_{i-1/2})(x - x_{i-1/2}) +$$

$$\frac{1}{2} g_{xx}(x_j, \theta_i)(x - x_{i-1/2})^2] dx. \tag{9.1.25}$$

The choice of $\xi_{i,1}$ and $\xi_{i,2}$ is now clear. We should select them so that

$$\int_{x_{i-1}}^{x_i} (x - \xi_{i,1})(x - \xi_{i,2}) dx = 0 \tag{9.1.26a}$$

and

$$\int_{x_{i-1}}^{x_i} (x - \xi_{i,1})(x - \xi_{i,2})(x - x_{i-1/2}) dx = 0. \tag{9.1.26b}$$

Assuming that this were possible, for the moment, then (9.1.25) would become

$$e_i(x_j) = \int_{x_{i-1}}^{x_i} \frac{1}{2}(x-\xi_{i,1})(x-\xi_{i,2})g_{xx}(x_j,\theta_i)(x-x_{i-1/2})^2]dx.$$

We bound this as

$$|e_i(x_j)| \le \frac{1}{2}\int_{x_{i-1}}^{x_i} |x-\xi_{i,1}||x-\xi_{i,2}||g_{xx}(x_j,\theta_i)||x-x_{i-1/2}|^2 dx$$

or

$$|e_i(x_j)| \le \frac{1}{2}h_i^5 \|g_{xx}(x_j,\cdot)\|_{i,\infty} \tag{9.1.27}$$

Substituting this result into (9.1.15) yields

$$|e(x_j)| \le \frac{1}{2}\sum_{i=1}^{N} h_i^5 \|g_{xx}(x_j,\cdot)\|_{i,\infty}, \qquad j=0,1,\ldots,N,$$

or

$$|e(x_j)| \le \frac{1}{2}\|g_{xx}(x_j,\cdot)\|_\infty \sum_{i=1}^{N} h_i^5 = \frac{1}{2}\|g_{xx}(x_j,\cdot)\|_\infty N h^5.$$

Thus,

$$|e(x_j)| \le Ch^4, \qquad j=0,1,\ldots,N, \tag{9.1.28a}$$

where

$$C = \frac{1}{2}(b-a)\|g_{xx}(x_j,\cdot)\|_\infty. \tag{9.1.28b}$$

The pointwise error has been increased by two orders with the special choice of collocation points dictated by (9.1.26). This is most definitely an example of nodal superconvergence since the global error is still $O(h^2)$.

It remains to determine the collocation points that satisfy (9.1.26). Let us begin by transforming these integrals to $[-1,1]$ using the mapping

$$x = x_{i-1/2} + \zeta\frac{h_i}{2}, \qquad -1 \le \zeta \le 1. \tag{9.1.29a}$$

Also let

$$\xi_{i,1} = x_{i-1/2} - \tau_1 h_i, \qquad \xi_{i,2} = x_{i-1/2} + \tau_2 h_i. \tag{9.1.29b}$$

Then

$$x - \xi_{i,1} = \frac{h_i}{2}(\zeta + 2\tau_1), \qquad x - \xi_{i,2} = \frac{h_i}{2}(\zeta - 2\tau_2). \tag{9.1.29c}$$

Conditions (9.1.26a,b) become

$$(\frac{h_i}{2})^3 \int_{-1}^{1} (\zeta + 2\tau_1)(\zeta - 2\tau_2) d\zeta = 0, \qquad (\frac{h_i}{2})^4 \int_{-1}^{1} (\zeta + 2\tau_1)(\zeta - 2\tau_2)\zeta d\zeta = 0.$$

Thus, it suffices to determine τ_1 and τ_2 such that

$$\int_{-1}^{1} (\zeta + 2\tau_1)(\zeta - 2\tau_2) P(\zeta) d\zeta = 0 \tag{9.1.30}$$

for all linear polynomials $P(\zeta)$.

Since the integrand is a cubic polynomial it can be evaluated exactly by the two-point Gauss-Legendre quadrature formula (cf. [5], Chapter 7)

$$\int_{-1}^{1} f(\zeta) d\zeta = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}) + E \tag{9.1.31a}$$

where the discretization error E is given by

$$E = \frac{1}{135} f^{iv}(\theta), \qquad \theta \in (-1, 1). \tag{9.1.31b}$$

Thus, applying (9.1.31) to (9.1.30), we have

$$(-\frac{1}{\sqrt{3}} + 2\tau_1)(-\frac{1}{\sqrt{3}} - 2\tau_2) P(-\frac{1}{\sqrt{3}}) + (\frac{1}{\sqrt{3}} + 2\tau_1)(\frac{1}{\sqrt{3}} - 2\tau_2) P(\frac{1}{\sqrt{3}}) = 0.$$

Once again, the integrand in (9.1.30) is a cubic polynomial so $E = 0$. We see that we can satisfy the above condition by choosing

$$\tau_1 = \tau_2 = \frac{1}{2\sqrt{3}}. \tag{9.1.32a}$$

Expressed in terms of the original variables through (9.1.29b), the collocation points are

$$\xi_{i,1} = x_{i-1/2} - \frac{h_i}{2\sqrt{3}}, \qquad \xi_{i,2} = x_{i-1/2} + \frac{h_i}{2\sqrt{3}}. \tag{9.1.32b}$$

Regardless of how the result is expressed, the key is to perform collocation at the Gauss-Legendre points mapped to the appropriate interval.

N	Midpoint Rule	Collocation
2		0.20×10^{-3}
5		0.64×10^{-5}
10	0.10×10^{-4}	0.46×10^{-6}
20	0.26×10^{-5}	0.33×10^{-7}
40	0.65×10^{-6}	0.23×10^{-8}
80	0.16×10^{-6}	0.16×10^{-9}

Table 9.1.1: Maximum pointwise errors in the solution of Example 9.1.1 using the midpoint rule and collocation at two Gauss-Legendre points.

The error formula (9.1.31b) can be used to obtain a more precise estimate of the collocation error than given by (9.1.28).

Let us conclude this section with two examples.

Example 9.1.1 (cf. [1], Chapter 5). Consider the problem

$$y'' + \frac{y'}{x} = (\frac{8}{8 - x^2})^2, \qquad 0 < x < 1, \qquad y'(0) = y(1) = 0,$$

which has the exact solution

$$y(x) = 2 \ln(\frac{7}{8 - x^2}).$$

The solution is smooth on $0 \leq x \leq 1$, but the coefficient $p(x) = 1/x$ is unbounded at $x = 0$. This would not be a problem when using collocation at the Gauss-Legendre points or, e.g., the midpoint rule since functions need not be evaluated at the endpoints. Formulas, such as the trapezoidal rule, would have to be modified to account for the singularity in $p(x)$. We'll avoid this and present results using both the midpoint rule and collocation (9.1.3) at the two Gauss-Legendre points (9.1.32). The maximum pointwise errors are presented in Table 9.1.1.

A simple calculation will verify that the midpoint rule and collocation solutions are converging at their expected rates of $O(N^{-2})$ and $O(N^{-4})$, respectively. The advantages of the higher-order collocation method on this problem are apparent.

Example 9.1.2 [4]. Consider

$$\epsilon y'' + [(1 + x)^2 y]' = \frac{e^{-x/2}}{2}[(1 + x)(3 - x) + \frac{\epsilon}{2}], \qquad 0 < x < 1,$$

$$y(0) = 0, \qquad y(1) = e^{-1/2} - e^{-7/3}\epsilon,$$

N	Error
4	0.44×10^{-4}
8	0.37×10^{-4}
16	0.58×10^{-8}
32	0.60×10^{-9}
64	0.19×10^{-10}
128	0.64×10^{-12}

Table 9.1.2: Maximum pointwise errors in the solution of Example 9.1.2 using collocation at Flaherty and Mathon's [4] points.

which has the exact solution

$$y(x) = e^{-x/2} - e^{-x(x^2+3x+3)/3\epsilon}.$$

For $0 < \epsilon \ll 1$ there is a boundary layer of width $O(\epsilon)$ at $x = 0$.

Collocation using cubic polynomials was performed at the Gauss-Legendre points and at the points

$$x_{i,1} = x_{i-1/2} - \tau h_i, \qquad x_{i,2} = x_{i-1/2} + \tau h_i,$$

where

$$\tau = \frac{1}{2} - \frac{2}{\rho_i} \frac{\omega(\rho_i/2)}{1 + \sqrt{1 - 4\omega(\rho_i/2)/\rho_i}},$$

$$\rho_i = \left| p(x_k) - \frac{q(x_k)}{p(x_k)} \right|, \qquad k = \begin{cases} i-1, & \text{if } \frac{p(x_{i-1})+p(x_i)}{2} < 0 \\ i, & \text{otherwise} \end{cases},$$

$$\omega(z) = \coth z - \frac{1}{z}.$$

With $\epsilon = 10^{-4}$ and $N = 8$, collocation at the Gauss-Legendre points produced a solution with a maximum pointwise error of approximately 15. The results using collocation at Flaherty and Mathon's [4] points are shown in Table 9.1.2.

Following the logic introduced in (9.1.30), Flaherty and Mathon selected their collocation points to reflect the boundary layer in the Green's function. Specifically, they chose points

$$\xi_{i,1} = x_{i-1/2} - \tau h_i, \qquad \xi_{i,2} = x_{i-1/2} + \tau h_i,$$

symmetrically disposed with respect to the center of each subinterval so that

$$\int_{-1}^{1} e^{-\frac{\rho(1\pm\zeta)}{2}} (\zeta^2 - 4\tau^2) P(\zeta) d\zeta = 0$$

for polynomials $P(\zeta)$ of as high a degree as possible.

9.2 Collocation for First-Order Systems

Let us extend the collocation methods to first-order vector BVPs of the usual form

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \qquad a < x < b, \tag{9.2.1a}$$

$$\mathbf{g}_L(\mathbf{y}(a)) = \mathbf{g}_R(\mathbf{y}(b)) = \mathbf{0}. \tag{9.2.1b}$$

We'll follow a different approach than the one used in Section 9.1 and integrate (9.2.1a) on a subinterval (x_{i-1}, x_i) to obtain

$$\mathbf{y}(x) = \mathbf{y}(x_{i-1}) + \int_{x_{i-1}}^{x} \mathbf{y}'(x)dx = \mathbf{y}(x_{i-1}) + \int_{x_{i-1}}^{x} \mathbf{f}(x, \mathbf{y})dx. \tag{9.2.2}$$

As with initial value methods, we'll construct a numerical method by approximating $\mathbf{y}'$ (or $\mathbf{f}$) by a polynomial and integrating the result. Any polynomial basis may be used, but let us concentrate on the Lagrange form of the interpolating polynomial

$$\mathbf{y}'(x) = \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) L_k(\zeta) + \mathbf{R}(\zeta) \tag{9.2.3a}$$

where

$$L_k(\zeta) = \prod_{j=1, j\neq k}^{J} \frac{\zeta - \tau_j}{\tau_k - \tau_j} = \frac{(\zeta - \tau_1)(\zeta - \tau_2)\ldots(\zeta - \tau_{k-1})(\zeta - \tau_{k+1})\ldots(\zeta - \tau_J)}{(\tau_k - \tau_1)(\tau_k - \tau_2)\ldots(\tau_k - \tau_{k-1})(\tau_k - \tau_{k+1})\ldots(\tau_k - \tau_J)}, \tag{9.2.3b}$$

$$\mathbf{R} = \mathbf{y}'[\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,J}] \prod_{j=1}^{J} (\zeta - \tau_j) \tag{9.2.3c}$$

$$x = x_{i-1} + \zeta h_i, \qquad 0 \leq \zeta \leq 1, \tag{9.2.3d}$$

$$\xi_{i,k} = x_{i-1} + \tau_k h_i. \tag{9.2.3e}$$

The image of the collocation points τ_k, $k = 1, 2, \ldots, J$, are ordered such that

$$0 \leq \tau_1 < \tau_2 < \cdots < \tau_J \leq 1. \tag{9.2.3f}$$

The divided difference $\mathbf{y}'[\xi_{i,1}, \xi_{i,2} \ldots, \xi_{i,J}]$ will be defined shortly.

Substituting (9.2.3a) into (9.2.2) yields

$$\mathbf{y}(x) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) \int_0^{\zeta} L_k(\eta) d\eta + h_i \int_0^{\zeta} \mathbf{R}(\eta) d\eta. \tag{9.2.4}$$

Evaluating (9.2.4) at the collocation points

$$\mathbf{y}(\xi_{ij}) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) \int_0^{\tau_j} L_k(\eta) d\eta + h_i \int_0^{\tau_j} \mathbf{R}(\eta) d\eta.$$

Let

$$a_{jk} = \int_0^{\tau_j} L_k(\eta) d\eta \tag{9.2.5a}$$

and

$$\mathbf{E}_j = \int_0^{\tau_j} \mathbf{R}(\eta) d\eta. \tag{9.2.5b}$$

Then

$$\mathbf{y}(\xi_{i,j}) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) a_{jk} + h_i \mathbf{E}_j. \tag{9.2.5c}$$

Evaluating (9.2.4) at $x = x_i$ yields

$$\mathbf{y}(x_i) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) \int_0^{1} L_k(\eta) d\eta + h_i \int_0^{1} \mathbf{R}(\eta) d\eta.$$

Letting

$$b_k = \int_0^{1} L_k(\eta) d\eta, \tag{9.2.6a}$$

and

$$\mathbf{E} = \int_0^{1} \mathbf{R}(\eta) d\eta. \tag{9.2.6b}$$

we have

$$\mathbf{y}(x_i) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{y}'(\xi_{i,k}) b_k + h_i \mathbf{E}. \tag{9.2.6c}$$

If the errors $\mathbf{E}_j$, $j = 1, 2, \ldots, J$, and $\mathbf{E}$ are neglected, we see that the collocation solution is obtained from a J stage implicit Runge-Kutta method. Thus, letting $\mathbf{Y}(x)$ denote the approximate solution, we have

$$\mathbf{Y}(x_i) = \mathbf{Y}(x_{i-1}) + h_i \sum_{k=1}^{J} b_k \mathbf{k}_{i,k} \tag{9.2.7a}$$

where

$$\mathbf{k}_{i,k} = \mathbf{Y}'(\xi_{i,k}) = \mathbf{f}(x_{i-1} + \tau_k h_i, \mathbf{Y}(x_{i-1}) + h_i \sum_{j=1}^{J} a_{kj} \mathbf{k}_{i,j}), \qquad k = 1, 2, \ldots, J. \tag{9.2.7b}$$

Solution continuity, required for first-order BVPs,

$$\mathbf{y}(x_i^-) = \mathbf{y}(x_i^+), \qquad i = 1, 2, \ldots, N-1,$$

is automatically satisfied.

Solutions at any point $x \in [x_{i-1}, x_i]$ are defined by the interpolation polynomial (9.2.4) and (9.2.3a) as

$$\mathbf{Y}(x) = \mathbf{Y}(x_{i-1}) + h_i \sum_{k=1}^{J} \mathbf{Y}'(\xi_{i,k}) \int_0^{\zeta} L_k(\eta) d\eta. \tag{9.2.7c}$$

Some common choices of the collocation points are:

1. *Gauss-Legendre points.* The points τ_k, $k = 1, 2, \ldots, J$, are the roots of the Legendre polynomial of degree J mapped to the interval $(0, 1)$. The roots of the Legendre polynomial are normally prescribed on $(-1, 1)$. This may be done by the linear transformation $\zeta = (1 + \sigma)/2$, $\sigma \in [-1, 1]$. For all J, $\tau_1 > 0$ and $\tau_J < 1$; thus, subinterval endpoints are not collocation points. The first five Legendre polynomials are shown in Table 9.2.1. The simplest scheme ($J = 1$) is the midpoint rule. The pointwise accuracy of a J-point scheme is $O(h^{2J})$.

2. *Radau points.* As described in Section 3.3, either $\tau_1 = 0$ or $\tau_J = 1$ and the remaining points τ_k, $k = 2, 3, \ldots J$, or $k = 1, 2, \ldots, J-1$, respectively, are selected to achieve maximal order. This involves selecting the points on $[-1, 1]$ as the roots of the Radau polynomial of degree J

$$R_J(\sigma) = P_J(\sigma) \mp \frac{J}{2J-1} P_{J-1}(\sigma)$$

J	$P_J(\sigma)$	$\sigma_j,\ j = 1, 2, \ldots, J$
0	1	
1	σ	0
2	$\frac{1}{2}(3\sigma^2 - 1)$	$\pm\frac{1}{\sqrt{3}}$
3	$\frac{\sigma}{2}(5\sigma^2 - 3)$	$0, \pm\sqrt{\frac{3}{5}}$
4	$\frac{1}{8}(35\sigma^4 - 30\sigma^2 + 3)$	$\pm 0.3399810436, \pm 0.8611363116$

Table 9.2.1: Legendre polynomials $P_J(\sigma)$ and their roots for $J = 0, 1, \ldots, 4$.

where $P_J(\sigma)$ is the Legendre polynomial of degree J. The negative sign is chosen with $\tau_J = 1$ and the positive sign is selected with $\tau_1 = 0$. The simplest scheme ($J = 1$) is the forward or backward Euler method when $\tau_1 = 0$ or $\tau_J = 1$, respectively. Judging from our experience with stiff IVPs, Radau schemes should be suitable for singularly perturbed BVPs. This will determine the proper choice of the fixed endpoint, as described in Chapter 10. The pointwise accuracy of a J point Radau scheme is $O(h^{2J-1})$.

3. *Lobatto points.* The points $\tau_1 = 0$, $\tau_J = 1$, and the remaining points τ_k, $k = 2, 3, \ldots, J-1$, are selected as the roots of the Legendre polynomial of degree $J-2$. The simplest scheme ($J = 2$) is the trapezoidal rule. The pointwise accuracy of a J point scheme is $O(h^{2(J-1)})$.

Example 9.1.1 ([1], Chapter 5). As in Example 9.1.1, consider

$$y'' + \frac{y'}{x} = (\frac{8}{8 - x^2})^2, \qquad 0 < x < 1, \qquad y'(0) = y(1) = 0.$$

Ascher *et al.* [1] solve this problem using

- two-point Gauss-Legendre collocation at

$$\tau_1 = \frac{1}{2}(1 - \frac{1}{\sqrt{3}}), \qquad \tau_2 = \frac{1}{2}(1 + \frac{1}{\sqrt{3}}),$$

- three-point Lobatto collocation at

$$\tau_1 = 0, \qquad \tau_2 = 1/2, \qquad \tau_3 = 1,$$

and

N	2-Point Gauss	3-Point Lobatto	3-Point Gauss
2	0.20×10^{-3}	0.17×10^{-4}	0.14×10^{-6}
5	0.64×10^{-5}	0.57×10^{-6}	0.70×10^{-9}
10	0.46×10^{-6}	0.37×10^{-7}	0.13×10^{-10}
20	0.33×10^{-7}	0.23×10^{-8}	0.27×10^{-12}
40	0.23×10^{-8}	0.15×10^{-9}	0.60×10^{-14}
80	0.16×10^{-9}	0.91×10^{-11}	0.13×10^{-14}

Table 9.2.2: Maximum pointwise errors in the solution of Example 9.2.1 using collocation at two Gauss-Legendre points, three Lobatto points, and three Gauss-Legendre points.

- three-point Gauss-Legendre collocation at

$$\tau_1 = \frac{1}{2}(1 - \sqrt{\frac{3}{5}}), \qquad \tau_2 = 1/2, \qquad \tau_3 = \frac{1}{2}(1 + \sqrt{\frac{3}{5}}).$$

The Gauss-Legendre methods do not need function evaluations at the endpoints of subintervals and, thus, the singular coefficient in the differential equation at $x = 0$ poses no problem. However, the Lobatto method must be modified to account for the singularity. Using L'Hopital's rule,

$$\lim_{x \to 0} \frac{y'(x)}{x} = y''(0).$$

Using this result in the differential equation yields $y''(0) = 1/2$; thus,

$$\lim_{x \to 0} \frac{y'(x)}{x} = \frac{1}{2}.$$

The maximum pointwise errors in y for the three methods are shown in Table 9.2.2. The two-point Gauss-Legendre, three-point Lobatto, and three-point Gauss-Legendre are converging at their expected rates of $O(N^{-4})$, $O(N^{-4})$, and $O(N^{-6})$, respectively.

The implementation of the collocation scheme is usually done by eliminating the unknowns $\mathbf{k}_{i,k}$, $k = 1, 2, \ldots, J$, appearing in (9.2.7b) on each subinterval and then solving for the nodal values $\mathbf{Y}_i = \mathbf{Y}(x_i)$, $i = 0, 1, \ldots, N$. We'll illustrate this for a linear system

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{A}(x)\mathbf{y} + \mathbf{b}(x), \tag{9.2.8a}$$

$$\mathbf{g}_L(\mathbf{y}(a)) = \mathbf{L}\mathbf{y}(a) - \mathbf{l}, \qquad \mathbf{g}_R(\mathbf{y}(b)) = \mathbf{R}\mathbf{y}(b) - \mathbf{r}. \tag{9.2.8b}$$

Nonlinear problems are solved using Newton's method to linearize them to the form of (9.2.8).

For a linear problem, (9.2.7b) becomes

$$\mathbf{k}_{i,k} = \mathbf{A}(\xi_{i,k})[\mathbf{Y}_{i-1} + h_i \sum_{j=1}^{J} a_{kj}\mathbf{k}_{i,j}] + \mathbf{b}(\xi_{i,k}), \qquad k = 1, 2, \ldots, J. \tag{9.2.9a}$$

This system can be written in matrix form as

$$\mathbf{W}_i\mathbf{k}_i = \mathbf{V}_i\mathbf{Y}_{i-1} + \mathbf{q}_i \tag{9.2.9b}$$

where

$$\mathbf{W}_i = \mathbf{I} - h_i \begin{bmatrix} a_{1,1}\mathbf{A}(\xi_{i,1}) & a_{1,2}\mathbf{A}(\xi_{i,1}) & & a_{1,J}\mathbf{A}(\xi_{i,1}) \\ a_{2,1}\mathbf{A}(\xi_{i,2}) & a_{2,2}\mathbf{A}(\xi_{i,2}) & & a_{2,J}\mathbf{A}(\xi_{i,2}) \\ \vdots & \vdots & \ddots & \vdots \\ a_{J,1}\mathbf{A}(\xi_{i,J}) & a_{J,2}\mathbf{A}(\xi_{i,J}) & & a_{J,J}\mathbf{A}(\xi_{i,J}) \end{bmatrix}, \tag{9.2.9c}$$

$$\mathbf{k}_i = \begin{bmatrix} \mathbf{k}_{i,1} \\ \mathbf{k}_{i,2} \\ \vdots \\ \mathbf{k}_{i,J} \end{bmatrix}, \qquad \mathbf{V}_i = \begin{bmatrix} \mathbf{A}(\xi_{i,1}) \\ \mathbf{A}(\xi_{i,2}) \\ \vdots \\ \mathbf{A}(\xi_{i,J}) \end{bmatrix}, \qquad \mathbf{q}_i = \begin{bmatrix} \mathbf{b}(\xi_{i,1}) \\ \mathbf{b}(\xi_{i,2}) \\ \vdots \\ \mathbf{b}(\xi_{i,J}) \end{bmatrix}. \tag{9.2.9d}$$

Let us write (9.2.7a) in the form

$$\mathbf{Y}_i = \mathbf{Y}_{i-1} + h_i\mathbf{D}\mathbf{k}_i \tag{9.2.10a}$$

where

$$\mathbf{D} = \begin{bmatrix} b_1\mathbf{I} & & & \\ & b_2\mathbf{I} & & \\ & & \ddots & \\ & & & b_J\mathbf{I} \end{bmatrix}. \tag{9.2.10b}$$

Eliminating $\mathbf{k}_i$ in (9.2.10a) using (9.2.9b) yields

$$\mathbf{Y}_i = \mathbf{\Gamma}_i\mathbf{Y}_{i-1} + \mathbf{g}_i, \qquad i = 1, 2, \ldots, J, \tag{9.2.11a}$$

where

$$\mathbf{\Gamma}_i = \mathbf{I} + h_i\mathbf{D}_i\mathbf{W}_i^{-1}\mathbf{V}_i, \tag{9.2.11b}$$

$$\mathbf{g}_i = h_i \mathbf{D}\mathbf{W}_i^{-1}\mathbf{q}_i. \tag{9.2.11c}$$

The resulting linear algebraic system is

$$\begin{bmatrix} \mathbf{L} & & & & \\ -\mathbf{\Gamma}_1 & \mathbf{I} & & & \\ & -\mathbf{\Gamma}_2 & & & \\ & & \ddots & & \\ & & & -\mathbf{\Gamma}_J & \mathbf{I} \\ & & & & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_0 \\ \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_J \end{bmatrix} = \begin{bmatrix} \mathbf{l} \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_J \\ \mathbf{r} \end{bmatrix}. \tag{9.2.11d}$$

Thus, once again, the algebraic system is block bidiagonal and may be solved by the methods of Section 8.5.

9.3 Convergence and Stability

Let us consider the linear first-order BVP

$$\mathcal{L}\mathbf{y} = \mathbf{y}' - \mathbf{A}(x)\mathbf{y} = \mathbf{b}(x), \qquad a < x < b, \tag{9.3.1a}$$

$$\mathbf{L}\mathbf{y}(a) = \mathbf{l}, \qquad \mathbf{R}\mathbf{y}(b) = \mathbf{r}. \tag{9.3.1b}$$

Using (9.2.4 - 9.2.6) we have

$$\mathbf{y}(\xi_{ij}) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} a_{jk}\mathbf{y}'(\xi_{i,k}) + h_i \mathbf{E}_j \tag{9.3.2a}$$

$$\mathbf{y}(x_i) = \mathbf{y}(x_{i-1}) + h_i \sum_{k=1}^{J} b_k \mathbf{y}'(\xi_{i,k}) + h_i \mathbf{E} \tag{9.3.2b}$$

where

$$\mathbf{E}_j = \int_0^{\tau_j} \mathbf{R}(\eta)d\eta \tag{9.3.2c}$$

and

$$\mathbf{E} = \int_0^1 \mathbf{R}(\eta)d\eta. \tag{9.3.2d}$$

The function $\mathbf{R}(x)$ is the interpolation error. In general, if we interpolate a function $f(x)$ at J distinct points $\{\xi_1 < \xi_2 < \cdots < \xi_J\}$ then

$$R(x) = f(x) - P(x) = f[\xi_1, \xi_2, \ldots, \xi_J] \prod_{j=1}^{J} (x - \xi_j) \tag{9.3.3}$$

where $P(x)$ is an interpolating polynomial having the form of (9.2.3a) (Section 5.2 or [5], Chapter 5). Recall, that the divided difference $f[\xi_1, \xi_2, \ldots, \xi_J]$ is defined recursively as

$$f[\xi_l, \xi_{l+1}, \ldots, \xi_{l+k}] = \begin{cases} f(\xi_l), & \text{if } k = 0 \\ \frac{f[\xi_{l+1}, \ldots, \xi_{l+k}] - f[\xi_l, \ldots, \xi_{l+k-1}]}{\xi_{l+k} - \xi_l}, & \text{if } k > 0 \end{cases} . \tag{9.3.4}$$

Recall (Lemma 5.2.1) that divided differences and derivatives are related in that there exists a point $\xi \in (\xi_1, \xi_J)$, $\xi_1 < \xi_2 < \cdots < \xi_J$, such that

$$f[\xi_l, \xi_{l+1}, \ldots, \xi_{l+k}] = \frac{f^{(J)}(\xi)}{J!} \tag{9.3.5}$$

when $f(x) \in C^J(\xi_1, \xi_J)$. We can specialize this result to the problem at hand.

Lemma 9.3.1. *Suppose* $\mathbf{y} \in C^{J+1}(a, b)$, *then the error (9.2.3c) in the interpolation (9.3.3) satisfies*

$$|\mathbf{R}(x)| = O(h_j^J), \qquad x \in (x_{j-1}, x_j), \qquad j = 1, 2, \ldots, J, \tag{9.3.6a}$$

where

$$h_j = x_j - x_{j-1} \tag{9.3.6b}$$

and $|\cdot|$ *denotes a vector norm.*

Proof. Using (9.2.3c)

$$\mathbf{R} = \mathbf{y}'[\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,J}] \prod_{j=1}^{J} (\zeta - \tau_j).$$

Since ζ and τ_j, $j = 1, 2, \ldots, J$, are on $[0, 1]$, the product appearing above has at most unit magnitude. Applying (9.3.5) in this case implies

$$\mathbf{y}'[\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,J}] = \frac{1}{J!} \frac{d^J}{d\zeta^J} \mathbf{y}'(\xi).$$

The notation is slightly confusing since $(\ )'$ denotes an x derivative and the remaining derivatives are taken with respect to ζ. We'll use (9.2.3d) to transfer all derivatives to the physical domain, i.e.,

$$\frac{d}{d\zeta} = h_j \frac{d}{dx}.$$

Then

$$\mathbf{y}'[\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,J}] = \frac{h_j^J}{J!}\mathbf{y}^{(J+1)}(\xi).$$

Taking a vector norm, we have

$$|\mathbf{R}(x)| \leq Ch_j^J$$

which proves the result. □

Having these preliminary results, we establish a basic stability result.

Theorem 9.3.1. *The J-stage collocation solution (9.2.7) of linear BVPs (9.2.8) exists and is stable. The discretization error*

$$\mathbf{e}(x) = \mathbf{y}(x) - \mathbf{y}(x) \tag{9.3.7a}$$

satisfies

$$\max_{x_{i-1} \leq x \leq x_i} \|\mathbf{e}^{(j)}(x)\| = O(h^{J-j})\theta_i^{j-1}, \qquad j = 0, 1, \ldots, J, \tag{9.3.7b}$$

where

$$\theta_i = \frac{h}{h_i}. \tag{9.3.7c}$$

Proof. Following the steps leading to (9.2.11), we use (9.3.2) to show that

$$\mathbf{y}(x_i) = \mathbf{\Gamma}_i \mathbf{y}(x_{i-1}) + \mathbf{g}_i + h_i \boldsymbol{\delta}_i. \tag{9.3.8}$$

The term $\boldsymbol{\delta}_i$ arises from the $\mathbf{E}$ and $\mathbf{E}_j$ terms in (9.3.2); hence, using (9.3.6) it is $O(h_i^J)$. If $J \geq 1$, the one-step scheme (9.2.7) is consistent. A consistent one-step scheme is also stable and convergent (Theorems 3.4.1, 2). Thus, solutions of (9.2.7) exist and, by subtacting (9.2.11a) from (9.3.8), satisfy

$$\max_{0 \leq i \leq N} |\mathbf{y}(x_i) - \mathbf{Y}_i| = O(h^J). \tag{9.3.9}$$

We can get more detailed information about errors at the collocation points by subtracting (9.2.7b) from (9.3.2a) while using (9.3.1a)

$$\mathbf{e}(\xi_{ij}) = \mathbf{e}(x_{i-1}) + h_i \sum_{k=1}^{J} a_{jk}\mathbf{A}(\xi_{i,k})\mathbf{e}(\xi_{i,k}) + h_i\mathbf{E}_j$$

Taking a matrix norm

$$|\mathbf{e}(\xi_{ij})| \le (1 + C_1 h_i)|\mathbf{e}(x_{i-1})| + C_2 h_i^{J+1}.$$

In obtaining this relationship, we bounded the Runge-Kutta coefficients and $|\mathbf{A}|$ by their maximal values and used consistency to infer that $\mathbf{e}(\xi_{i,k})$ does not differ from $\mathbf{e}(x_{i-1})$ by more than $O(h_i)$. The last term above was bounded using (9.3.6).

Since (9.3.9) implies that $|\mathbf{e}(x_j)| = O(h^J)$, $j = 0, 1, \ldots, J$, we have

$$\max_{0\le i\le N, 1\le j\le J} |\mathbf{e}(\xi_{ij})| = O(h^J). \tag{9.3.10}$$

Furthermore, since the exact and numerical solutions satisfy the differential equation (9.3.1) at the collocation points, we have

$$\mathbf{e}'(\xi_{ij}) = \mathbf{A}(\xi_{ij})\mathbf{e}(\xi_{ij}).$$

Taking a vector norm and using (9.3.10)

$$\max_{0\le i\le N, 1\le j\le J} |\mathbf{e}'(\xi_{ij})| = O(h^J). \tag{9.3.11}$$

Finally, applying the interpolation formula (3) to $\mathbf{y}'$ and using (9.2.3a), we have

$$\mathbf{e}'(x) = \sum_{k=1}^{J} \mathbf{e}'(\xi_{i,k})L_k(\zeta) + \mathbf{R}(\zeta). \tag{9.3.12}$$

Differentiating and using (9.3.6) yields the result (9.3.7b). □

This result is not as sharp as it could be as described by the next Theorem.

Theorem 9.3.2. *Suppose that the collocation points are distinct with* $\xi_{i,1} < \xi_{i,2} < \cdots < \xi_{i,J}$, $i = 1, 2, \ldots, N$, *the one-step method (9.2.7b) is accurate to* $O(h^p)$, *and* $\mathbf{A}(x), \mathbf{b}(x) \in C^p(a, b)$. *Then*

$$|\mathbf{Y}(x_i) - \mathbf{y}(x_i)| = O(h^p), \qquad i = 0, 1, \ldots, N, \tag{9.3.13a}$$

$$|\mathbf{Y}(\xi_{ij}) - \mathbf{y}(\xi_{ij})| = O(h_i^{J+1}) + O(h^p), \qquad j = 1, 2, \ldots, J, \qquad i = 0, 1, \ldots, N. \tag{9.3.13b}$$

Remark. If $p > J$ convergence at the nodes and collocation points is at a higher rate than implied by Theorem 9.3.1. This is the phenomenon of superconvergence that we illustrated in Section 9.1.

Proof. As in Section 9.1, introduce the Green's function

$$\mathbf{e}(\xi) = \int_a^b G(\xi, x)\mathcal{L}\mathbf{e}(x)dx = \sum_{j=1}^{J} \int_{x_{j-1}}^{x_j} G(\xi, x)\mathcal{L}\mathbf{e}(x)dx. \tag{9.3.14}$$

Let us assume that $\xi \notin (x_{j-1}, x_j)$ and, following the logic introduced in Section 9.1, write

$$G(\xi, x)\mathcal{L}\mathbf{e}(x) = \mathbf{w}(x)\prod_{j=1}^{J}(x - \xi_{ij})$$

The function $\mathbf{w}(x)$ involves the J *th* derivative of $\mathcal{L}\mathbf{e}$. Using this and (9.3.7b), we may expect $\mathbf{w}(x)$ to have $p - J$ bounded derivatives. Thus, expand $\mathbf{w}(x)$ in a Taylor's series of the form

$$\mathbf{w}(x) = \mathbf{P}_{p-J-1}(x) + O(h_i^{p-J})$$

where $\mathbf{P}_{p-J-1}(x)$ is a polynomial of degree $p - J - 1$.

If the one-step method is accurate to order p then

$$\int_{x_{j-1}}^{x_j} \mathbf{P}_{p-J-1}(x)\prod_{j=1}^{J}(x - \xi_{ij})dx = 0.$$

Since

$$\prod_{j=1}^{J}(x - \xi_{ij}) = O(h_i^J)$$

we have

$$\int_{x_{j-1}}^{x_j} G(\xi, x)\mathcal{L}\mathbf{e}(x)dx = O(h_i^{P-J})O(h_i^J)h_i, \qquad \xi \notin (x_{j-1}, x_j). \tag{9.3.15a}$$

The result (9.3.13a) is obtained by summing the above relation over the subintervals.

When $\xi \in (x_{j-1}, x_j)$ then we are only able to show that

$$\int_{x_{j-1}}^{x_j} G(\xi, x)\mathcal{L}\mathbf{e}(x)dx = O(h_i^{J+1}), \qquad \xi \in (x_{j-1}, x_j). \tag{9.3.15b}$$

Summing (9.3.15a,b) yields (9.3.13b). □

Superconvergence occurs whenever $p > J + 1$.

Bibliography

[1] U.M. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM, Philadelphia, second edition, 1995.

[2] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[3] C. de Boor and B. Swartz. Collocation at gaussian points. *SIAM J. Numer. Anal.*, 10:582–687, 1973.

[4] J.E. Flaherty and W. Mathon. Collocation with polynomial and tension splines for singularly perturbed boundary value problems. *SIAM J. Sci. Stat. Comput*, 1:260–289, 1980.

[5] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. John Wiley and Sons, New York, 1966.