CHAPTER SEVEN

# Numerical Methods in Dynamics

The equations of motion for constrained multibody systems derived in Chapter 6 must be assembled and solved numerically. Three distinct modes of dynamic analysis may be carried out: (1) equilibrium, (2) inverse dynamic, and (3) dynamic. Section 7.1 presents an overview of the computations required to carry out each of the three modes of analysis. The properties of mixed differential–algebraic equations of motion are briefly studied in Section 7.2 to provide the theoretical foundation needed for numerical solution. Three distinct methods for the solution of mixed differential–algebraic equations are presented in Section 7.3, and a hybrid method that takes advantage of favorable properties of each is presented. The result is a numerical reduction of the mixed differential–algebraic equations to a system of first-order differential equations that can be integrated using the standard numerical integration algorithms outlined in Section 7.4. Finally, a method for the minimization of total potential energy is presented in Section 7.4 for equilibrium analysis.

The theory and numerical methods for integration of initial-value problems of ordinary differential equations is well developed and documented in textbooks [31, 36, 37] and computer codes. A review of computer codes that are applicable for integrating differential equations of dynamics may be found in Reference 38. The presentation of methods in this chapter is only a brief introduction to the subject.

It is fascinating that, even though the Lagrange multiplier form of the equations of motion for constrained mechanical systems has been known since the late 1700s, only in 1981 was it fully recognized [39] that these equations cannot be treated as differential equations. A flurry of research activity on methods of integrating mixed differential–algebraic equations of dynamics has occurred in the 1980s [19, 40–43]. Presentation of methods for integrating differential–algebraic equations of dynamics in this text is limited to algorithms that are specialized for this field. It is anticipated that developments in the late 1980s will yield more generally applicable algorithms and computer codes.

## 7.1 ORGANIZATION OF COMPUTATIONS

The DADS computer code introduced in Section 4.1 for kinematic analysis also carries out dynamic analysis using the theory presented in Chapter 6 and numerical methods presented in the following sections of this chapter. Many of the organizational aspects of computation discussed in Chapter 4 for kinematic analysis are also used in DADS for dynamic analysis, so they will not be discussed here in detail. This section focuses on the organization of computations for dynamic analysis.

Especially for large-scale dynamics applications, bookkeeping and equation formulation tasks are extensive. As shown by even the simple examples analyzed in Chapter 6, the dimension of constrained equations of motion for multibody systems is about twice that encountered in kinematic analysis and the equations are nonlinear. Furthermore, solution of the equations of dynamics involves the integration of mixed differential–algebraic equations, rather than solution of algebraic equations in kinematic analysis. Automation of computation for multibody system dynamic analysis is required to obtain a general-purpose applications software capability.

DADS computational flow to implement dynamic analysis is summarized in Fig. 7.1.1. For purposes of dynamic analysis, the DADS code employs three basic program segments: (1) a preprocessor that assembles problem definition information and organizes data for computation, (2) a dynamic analysis program that constructs and solves the equations of dynamics, and (3) a postprocessor that prepares output information and displays the results of a dynamic simulation. Many of the subroutines employed for dynamic analysis are also used in kinematic analysis.

As indicated in Fig. 7.1.1, in addition to kinematic data, which are identical to those required for the kinematic analysis in Fig. 4.1.1, inertia, force, and initial condition data are required for dynamic analysis. The analysis mode (equilibrium, dynamic, or inverse dynamic) and output data desired must also be specified. The output of preprocessor computation is a dynamic analysis data set that is read and implemented by the dynamic analysis program.

Prior to dynamic analysis, the system is assembled and checked for feasibility, just as in kinematic analysis. During dynamic analysis, mass-, constraint-, and force-related matrices are assembled and used for either equilibrium analysis, transient dynamic analysis, or inverse dynamic analysis. The nature of the computations carried out during each of these three modes of analysis is quite different. Equilibrium analysis may be accomplished through either dynamic settling or minimization of total potential energy. Dynamic analysis is carried out by numerically integrating mixed differential–algebraic equations of motion. Finally, inverse dynamic analysis involves algebraic solution of the equations of kinematics and subsequent algebraic solution for Lagrange multipliers and forces that act in the kinematically driven system.

Following completion of dynamic analysis, the DADS postprocessor or-

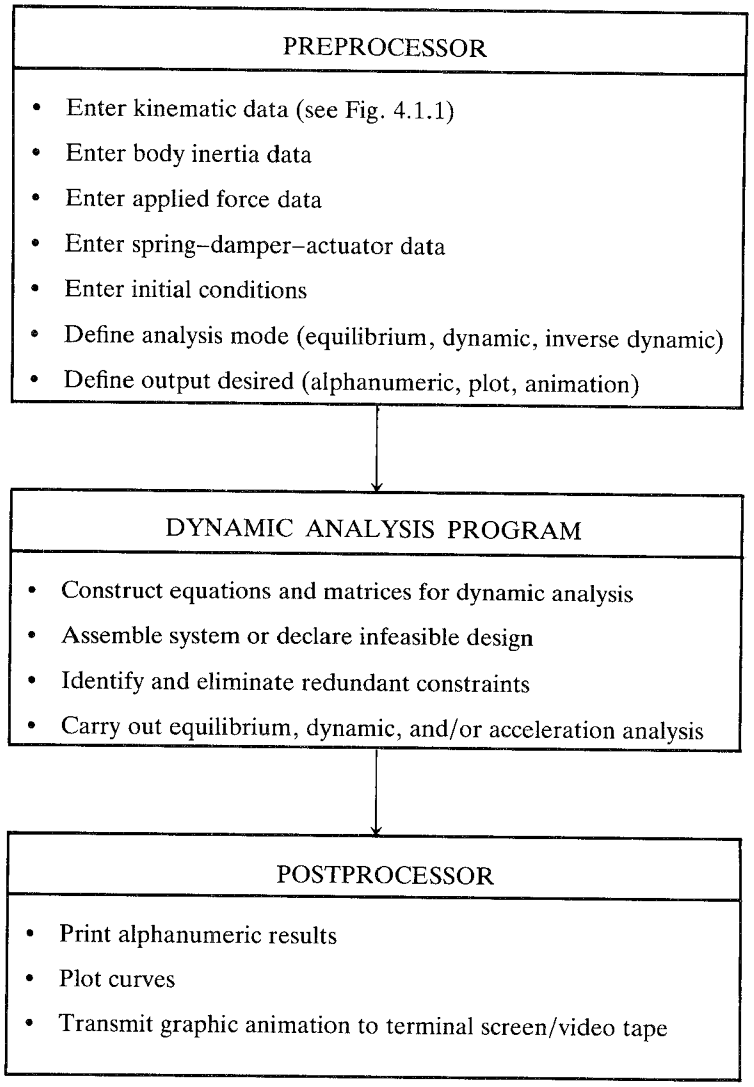
PREPROCESSOR
• Enter kinematic data (see Fig. 4.1.1)
• Enter body inertia data
• Enter applied force data
• Enter spring–damper–actuator data
• Enter initial conditions
• Define analysis mode (equilibrium, dynamic, inverse dynamic)
• Define output desired (alphanumeric, plot, animation)
DYNAMIC ANALYSIS PROGRAM
• Construct equations and matrices for dynamic analysis
• Assemble system or declare infeasible design
• Identify and eliminate redundant constraints
• Carry out equilibrium, dynamic, and/or acceleration analysis
POSTPROCESSOR
• Print alphanumeric results
• Plot curves
• Transmit graphic animation to terminal screen/video tape

**Figure 7.1.1** DADS dynamics computational flow.

ganizes and transmits the results of the simulation to a printer, plotter, or animation workstation.

As in the case of kinematic analysis, implementation of the extensive logical and numerical computations that are identified by the computational flow in Fig. 7.1.1 requires a large-scale computer code, the details of which are beyond the scope of this text. Prior to delving into the numerical methods that are used to carry out each of the modes of dynamic analysis, it is of value to understand the

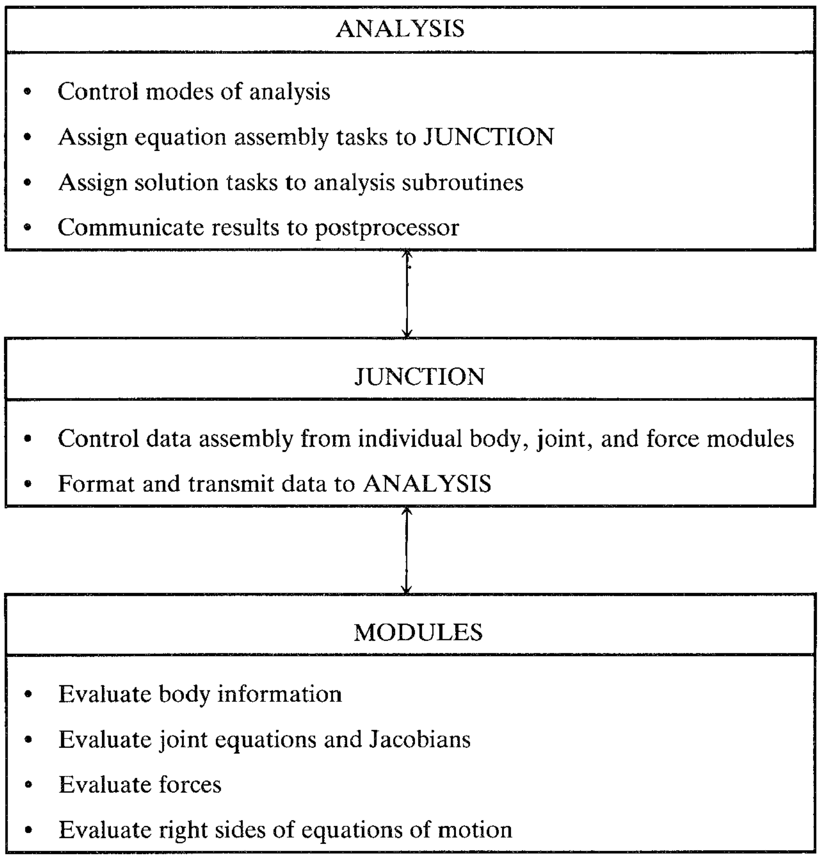
ANALYSIS
• Control modes of analysis
• Assign equation assembly tasks to JUNCTION
• Assign solution tasks to analysis subroutines
• Communicate results to postprocessor
JUNCTION
• Control data assembly from individual body, joint, and force modules
• Format and transmit data to ANALYSIS
MODULES
• Evaluate body information
• Evaluate joint equations and Jacobians
• Evaluate forces
• Evaluate right sides of equations of motion

**Figure 7.1.2** Structure of DADS dynamic analysis program.

flow of information that must be generated during dynamic analysis. The structure of the DADS dynamic analysis program is illustrated schematically in Fig. 7.1.2. The analysis program defines control over the modes of analysis and assigns equation assembly tasks to the junction program, which in turn calls modules that generate the required information and transmit it to the analysis program. Once equations are generated at each time step, the analysis program assigns solution tasks to analysis subroutines and communicates results to the postprocessor. The same modules used in kinematic analysis are also used in dynamic analysis, but they generate additional data that are required for dynamic analysis. In addition, force element modules generate force data that are required in the equations of motion. Rather than having separate programs for kinematic and dynamic analysis, the functions required for both are embedded in common modules, and only the functions required to support the analysis being undertaken are exercised.

A more detailed definition of computational flow and information that is generated during dynamic analysis are presented in Fig. 7.1.3. Input data reading and problem setup functions and model assembly and feasibility analysis functions

| ANALYSIS | | JUNCTION/MODULES |
|---|---|---|
| **Input Data Reading and Setup** | | |
| • Read data<br>• Assign module tasks<br>• Assign array dimensions/addresses | $\longleftrightarrow$ | • Count number of generalized coordinates and Lagrange multipliers<br>• Assign addresses of nonzero matrix entries |
| **Model Assembly and Feasibility Analysis** | | |
| • Assign module tasks<br>• Assemble using minimization computation<br>• Identify redundant constraints | $\longleftrightarrow$ | Evaluate constraint equations<br>Evaluate constraint Jacobians |
| **Equilibrium Analysis** | | |
| • Assign module tasks<br>• Minimize total potential energy | $\longleftrightarrow$ | • Evaluate constraint Jacobians<br>• Evaluate forces ~~and computation~~ |
| **Dynamic Analysis** | | |
| • Assign module tasks<br>• Solve for acceleration<br>• Integrate for position and velocity | $\longleftrightarrow$ | • Evaluate constraint Jacobians and mass matrices<br>• Evaluate forces and equation right sides |
| **Inverse Dynamic Analysis** | | |
| • Assign module tasks<br>• Solve kinematic equations<br>• Evaluate Lagrange multipliers and forces | $\longleftrightarrow$ | • Evaluate kinematic equations<br>• Evaluate equations of motion |

**Figure 7.1.3** DADS dynamic analysis flow.

are identical to those encountered in kinematic analysis. Dimensioning and addressing of arrays, of course, requires consideration of new variables, such as Lagrange multipliers, that must be determined during dynamic analysis.

Equilibrium analysis is carried out either by direct application of dynamic analysis for dynamic settling or minimization of total potential energy for conservative mechanical systems. Dynamic analysis is carried out using a hybrid method of reducing mixed differential–algebraic equations to differential equations and subsequent numerical integration for position and velocity of the system throughout the time interval of interest. Finally, inverse dynamic analysis is carried out by solving kinematic equations for a kinematically determined system, assembling the equations of motion, and solving for Lagrange multipliers. The reaction and driving forces and torques that are required to impose the specified motion are subsequently calculated.

## 7.2 SOLUTION OF MIXED DIFFERENTIAL–ALGEBRAIC EQUATIONS OF MOTION

The equations of motion derived in Chapter 6 are summarized in matrix form as Eq. 6.3.18, which is repeated here as

$$\begin{bmatrix} \mathbf{M} & \mathbf{\Phi}_{\mathbf{q}}^T \\ \mathbf{\Phi}_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}^A \\ \boldsymbol{\gamma} \end{bmatrix} \tag{7.2.1}$$

In the case of planar systems, the mass matrix $\mathbf{M}$ is constant and $\mathbf{Q}^A$ is the generalized applied force. Precisely the same form of spatial equations of motion is obtained in Chapter 11, but with a mass matrix that may depend on generalized coordinates and nonlinear terms in velocities and generalized coordinates that represent Coriolis acceleration effects on the right side. For the numerical solution techniques presented in this chapter to be applicable in the general case, it will be presumed that the mass matrix $\mathbf{M}$ and generalized applied force vector $\mathbf{Q}^A$ are nonlinear functions of generalized coordinates, their velocities, and time.

In addition to the equations of motion in Eq. 7.2.1, the kinematic constraints of Eq. 6.3.4 must hold, repeated here as

$$\mathbf{\Phi}(\mathbf{q}, t) = \mathbf{0} \tag{7.2.2}$$

where $\mathbf{\Phi}(\mathbf{q}, t)$ is presumed to have two continuous derivatives with respect to its arguments. In addition, the velocity equations of Eq. 6.3.17 must hold, repeated here as

$$\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} = \boldsymbol{\nu} \tag{7.2.3}$$

The differential and algebraic equations of Eqs. 7.2.1 to 7.2.3 comprise the *mixed differential–algebraic equations of motion.*

In addition to the position and velocity equations of Eqs. 7.2.2 and 7.2.3, which must hold at the initial time $t_0$, additional initial conditions on position and velocity must be introduced, as in Eqs. 6.3.26 and 6.3.27, repeated here as

$$\mathbf{\Phi}^I(\mathbf{q}(t_0), t_0) = \mathbf{0} \tag{7.2.4}$$

$$\mathbf{B}^I\dot{\mathbf{q}}(t_0) = \mathbf{v}^I \tag{7.2.5}$$

It is essential that the initial position condition of Eqs. 7.2.4 and 7.2.2, evaluated at $t_0$, uniquely determine the initial position $\mathbf{q}(t_0)$. Similarly, it is essential that the initial velocity conditions of Eqs. 7.2.5 and 7.2.3, evaluated at $t_0$, uniquely determine the initial velocity $\dot{\mathbf{q}}(t_0)$.

As shown in Section 6.3, the coefficient matrix in Eq. 7.2.1 is nonsingular for meaningful physical systems. Therefore, it may theoretically be inverted to obtain

$$\begin{aligned}\ddot{\mathbf{q}} &= \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, t)\\ \boldsymbol{\lambda} &= \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t)\end{aligned} \tag{7.2.6}$$

where, by the implicit function theorem, the vector functions on the right are continuously differentiable with respect to their arguments. This result is theoretical, since explicit inversion of a large coefficient matrix with variable terms is impractical. Nevertheless, this theoretical result shows that $\ddot{\mathbf{q}}$ may be uniquely determined.

Direct numerical integration for $\mathbf{q}$ and $\dot{\mathbf{q}}$, from Eq. 7.2.6 and initial conditions, is both theoretically and computationally nontrivial. Since $\mathbf{q}$ and $\dot{\mathbf{q}}$ must satisfy Eqs. 7.2.2 and 7.2.3, all components of $\mathbf{q}$ and $\dot{\mathbf{q}}$ are not independent. One algorithm for integration that is used in some computer programs is to ignore this dependence and simply integrate accelerations that are obtained by numerically solving Eq. 7.2.1. There is no guarantee, however, that the function $\mathbf{q}(t)$ obtained will accurately satisfy the constraints of Eqs. 7.2.2 and 7.2.3. This difficulty arises because the equations of motion are in fact mixed differential–algebraic equations.

To obtain insight into the differential–algebraic equations of motion, consider a virtual displacement $\delta\mathbf{q}$ that satisfies the constraint equations to first order; that is, $\mathbf{\Phi}_{\mathbf{q}}\,\delta\mathbf{q} = \mathbf{0}$. At a value of $\mathbf{q}$ that satisfies the constraints of Eq. 7.2.2, the Gaussian reduction technique of Section 4.4 may be employed to transform this equation to the form of Eq. 4.4.8, that is,

$$\mathbf{U}\,\delta\mathbf{u} + \mathbf{R}\,\delta\mathbf{v} = \mathbf{0} \tag{7.2.7}$$

Since the coefficient matrix $\mathbf{U}$ is triangular with unit values on the diagonal, it is nonsingular, and Eq. 7.2.7 uniquely determines $\delta\mathbf{u}$ once values are selected for $\delta\mathbf{v}$. Therefore, $\mathbf{v}$ is interpreted as a vector of *independent generalized coordinates* and $\mathbf{u}$ is a vector of *dependent generalized coordinates*.

Since $\mathbf{U}$ in Eq. 7.2.7 is a matrix that is obtained by elementary row operations [22] on the matrix $\mathbf{\Phi}_{\mathbf{u}}$, $\mathbf{\Phi}_{\mathbf{u}}$ is nonsingular. Therefore, by the implicit function theorem, Eq. 7.2.2 can theoretically be solved for $\mathbf{u}$ as a function of $\mathbf{v}$

and $t$; that is,

$$\mathbf{u} = \mathbf{h}(\mathbf{v}, t) \tag{7.2.8}$$

where the vector function $\mathbf{h}(\mathbf{v}, t)$ is twice continuously differentiable in its arguments. Since Eq. 7.2.2 is highly nonlinear, explicit construction of the functional relationship in Eq. 7.2.8 is impractical. The result, however, is of great theoretical importance in analyzing the nature of solutions of mixed differential–algebraic equations of motion.

To analyze the equations of motion of Eq. 7.2.1, they may be reordered and partitioned, according to the decomposition of $\mathbf{q}$ into $\mathbf{u}$ and $\mathbf{v}$, as

$$\begin{gathered}\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}} + \mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}} + \mathbf{\Phi}_{\mathbf{u}}^T\boldsymbol{\lambda} = \mathbf{Q}^{A\mathbf{u}} \\ \mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}} + \mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}} + \mathbf{\Phi}_{\mathbf{v}}^T\boldsymbol{\lambda} = \mathbf{Q}^{A\mathbf{v}} \\ \mathbf{\Phi}_{\mathbf{u}}\ddot{\mathbf{u}} + \mathbf{\Phi}_{\mathbf{v}}\ddot{\mathbf{v}} = \boldsymbol{\gamma}\end{gathered} \tag{7.2.9}$$

where the mass matrices in the first two equations are submatrices of $\mathbf{M}$, and the vector functions on the right are a partitioning of the generalized applied force vector $\mathbf{Q}^A$. Similarly, the velocity equation of Eq. 7.2.3 may be written in the form

$$\mathbf{\Phi}_{\mathbf{u}}\dot{\mathbf{u}} + \mathbf{\Phi}_{\mathbf{v}}\dot{\mathbf{v}} = \boldsymbol{\nu} \tag{7.2.10}$$

For purposes of theoretical analysis, Eq. 7.2.8 may be used to write all expressions that involve the dependent generalized coordinates $\mathbf{u}$ as functions of the independent coordinates $\mathbf{v}$ and time. Therefore, all terms in Eqs. 7.2.9 and 7.2.10 that depend on $\mathbf{u}$ may be interpreted as functions of $\mathbf{v}$. Since the coefficient matrix of $\dot{\mathbf{u}}$ in Eq. 7.2.10 is nonsingular, the dependent velocity $\dot{\mathbf{u}}$ may theoretically be written as

$$\dot{\mathbf{u}} = \mathbf{\Phi}_{\mathbf{u}}^{-1}[\boldsymbol{\nu} - \mathbf{\Phi}_{\mathbf{v}}\dot{\mathbf{v}}] \tag{7.2.11}$$

which, after using Eq. 7.2.8, may be written as a function of only $\mathbf{v}$ and $\dot{\mathbf{v}}$. Similarly, the third of Eqs. 7.2.9 may theoretically be solved for $\ddot{\mathbf{u}}$, to obtain

$$\ddot{\mathbf{u}} = \mathbf{\Phi}_{\mathbf{u}}^{-1}[\boldsymbol{\gamma} - \mathbf{\Phi}_{\mathbf{v}}\ddot{\mathbf{v}}] \tag{7.2.12}$$

which, after using Eqs. 7.2.8 and 7.2.11, may be written in terms of $\mathbf{v}$, $\dot{\mathbf{v}}$, and $\ddot{\mathbf{v}}$.

Since the coefficient matrix of $\boldsymbol{\lambda}$ in the first of Eqs. 7.2.9 is nonsingular, this equation may theoretically be solved for $\boldsymbol{\lambda}$ to obtain

$$\boldsymbol{\lambda} = (\mathbf{\Phi}_{\mathbf{u}}^{-1})^T[\mathbf{Q}^{A\mathbf{u}}(\mathbf{v}, \dot{\mathbf{v}}, t) - \mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}} - \mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}}] \tag{7.2.13}$$

Substituting this result and Eq. 7.2.12 into the second of Eqs. 7.2.9 yields (Prob. 7.2.1)

$$\hat{\mathbf{M}}^{\mathbf{v}}(\mathbf{v}, \dot{\mathbf{v}}, t)\ddot{\mathbf{v}} = \hat{\mathbf{Q}}^{\mathbf{v}}(\mathbf{v}, \dot{\mathbf{v}}, t) \tag{7.2.14}$$

where

$$\hat{\mathbf{M}}^{\mathbf{v}} = \mathbf{M}^{\mathbf{vv}} - \mathbf{M}^{\mathbf{vu}}\mathbf{\Phi}_{\mathbf{u}}^{-1}\mathbf{\Phi}_{\mathbf{v}} - \mathbf{\Phi}_{\mathbf{v}}^T(\mathbf{\Phi}_{\mathbf{u}}^{-1})^T[\mathbf{M}^{\mathbf{uv}} - \mathbf{M}^{\mathbf{uu}}\mathbf{\Phi}_{\mathbf{u}}^{-1}\mathbf{\Phi}_{\mathbf{v}}] \tag{7.2.15}$$

$$\hat{\mathbf{Q}}^{\mathbf{v}} = \mathbf{Q}^{A\mathbf{v}} - \mathbf{M}^{\mathbf{vu}}\mathbf{\Phi}_{\mathbf{u}}^{-1}\boldsymbol{\gamma} - \mathbf{\Phi}_{\mathbf{v}}^T(\mathbf{\Phi}_{\mathbf{u}}^{-1})^T[\mathbf{Q}^{A\mathbf{u}} - \mathbf{M}^{\mathbf{uu}}\mathbf{\Phi}_{\mathbf{u}}^{-1}\boldsymbol{\gamma}] \tag{7.2.16}$$

Equation 7.2.14 is a set of differential equations in only the independent generalized coordinates $\mathbf{v}$ that are consistent with the position and velocity constraints that act on the system.

---

**Example 7.2.1:** To illustrate the foregoing theoretical reduction process, consider the simple pendulum of Example 6.3.1, with $\ell = 1$. For this elementary system, the constraint equations are

$$\mathbf{\Phi}(\mathbf{q}) = \begin{bmatrix} x_1 - \cos\phi_1 \\ y_1 - \sin\phi_1 \end{bmatrix} = \mathbf{0}$$

The constraint Jacobian is

$$\mathbf{\Phi}_\mathbf{q} = \begin{bmatrix} 1 & 0 & \sin\phi_1 \\ 0 & 1 & -\cos\phi_1 \end{bmatrix}$$

and virtual displacements $\delta\mathbf{q}$ satisfy

$$\mathbf{I}\begin{bmatrix} \delta x_1 \\ \delta y_1 \end{bmatrix} + \begin{bmatrix} \sin\phi_1 \\ -\cos\phi_1 \end{bmatrix}\delta\phi_1 = \mathbf{0}$$

which is of the form of Eq. 7.2.7, with $\mathbf{u} = [x_1, y_1]^T$ and $v = \phi_1$. Thus,

$$\mathbf{\Phi}_\mathbf{u} = \mathbf{I}$$

$$\mathbf{\Phi}_v = \begin{bmatrix} \sin\phi_1 \\ -\cos\phi_1 \end{bmatrix}$$

In this elementary example, the constraint equations may be trivially solved for $\mathbf{u}$ as

$$\mathbf{u} = \begin{bmatrix} \cos\phi_1 \\ \sin\phi_1 \end{bmatrix} \equiv \mathbf{h}(v)$$

The right sides of the velocity and acceleration equations are $\mathbf{v} = \mathbf{0}$ and

$$\boldsymbol{\gamma} = \begin{bmatrix} -\cos\phi_1 \\ -\sin\phi_1 \end{bmatrix}\dot{\phi}_1^2$$

Thus, Eqs. 7.2.11 and 7.2.12 are

$$\dot{\mathbf{u}} = \begin{bmatrix} -\sin\phi_1 \\ \cos\phi_1 \end{bmatrix}\dot{v}$$

$$\ddot{\mathbf{u}} = \begin{bmatrix} -\cos\phi_1 \\ -\sin\phi_1 \end{bmatrix}\dot{v}^2 - \begin{bmatrix} \sin\phi_1 \\ -\cos\phi_1 \end{bmatrix}\ddot{v}$$

The partitioned form of the equations of motion of Eq. 7.2.9 is (Prob. 7.2.2)

$$\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix}\ddot{\mathbf{u}} + \boldsymbol{\lambda} = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

$$J'\ddot{v} + [\sin\phi_1, -\cos\phi_1]\boldsymbol{\lambda} = 0$$

Carrying out the reduction that leads to Eqs. 7.2.14 through 7.2.16 (or simply evaluating terms in Eqs. 7.2.15 and 7.2.16) yields

$$\hat{M}^v = J' + m$$
$$\hat{Q}^v = -mg\cos v$$

Thus, the reduced differential equation of motion of Eq. 7.2.14 is

$$(J' + m)\ddot{v} = -mg\cos v$$

---

The reader is cautioned that, while the theoretical reduction presented in this section can be carried out explicitly for the simple pendulum, it is not practical for more realistic systems. Even in the case of the two-body model of a slider–crank mechanism of Example 6.3.2, this reduction procedure is impractical.

To assure the existence of solutions of Eq. 7.2.14, it is important to show that the coefficient matrix on the left is nonsingular. To see that this is true, first observe that *kinematically admissible virtual velocities* $\delta\dot{\mathbf{q}} = [\delta\dot{\mathbf{u}}^T, \delta\dot{\mathbf{v}}^T]^T$ satisfy the velocity equation, with time held fixed, that is;

$$\mathbf{\Phi}_\mathbf{u}\,\delta\dot{\mathbf{u}} + \mathbf{\Phi}_\mathbf{v}\,\delta\dot{\mathbf{v}} = \mathbf{0} \tag{7.2.17}$$

which is essentially Eq. 7.2.10 with the right side equal to zero. Using the definition of the reduced mass matrix of Eqs. 7.2.15 and 7.2.17, the quadratic form associated with kinetic energy and independent virtual velocities may be expanded to obtain (Prob. 7.2.3)

$$\delta\dot{\mathbf{v}}^T\hat{\mathbf{M}}^\mathbf{v}\,\delta\dot{\mathbf{v}} = \delta\dot{\mathbf{q}}^T\mathbf{M}\,\delta\dot{\mathbf{q}} > 0 \tag{7.2.18}$$

which must hold for all nonzero independent virtual velocities $\delta\dot{\mathbf{v}}$. Since independent virtual velocities are arbitrary, the matrix of Eq. 7.2.15 is positive definite, and hence nonsingular. From well-known theory of ordinary differential equations [44], the differential equation of Eq. 7.2.14 and initial conditions on $\mathbf{v}$ defined by Eqs. 7.2.2 through 7.2.5 have a unique solution $\mathbf{v}(t)$. This $\mathbf{v}(t)$ and the associated $\mathbf{u}(t)$ defined by Eq. 7.2.8 satisfy the mixed differential–algebraic equations of motion (Eqs. 7.2.1 through 7.2.3).

The preceding argument guarantees the existence of a unique solution of the mixed differential–algebraic equations of motion under modest hypotheses on the kinematic and kinetic structure of the system. It shows that loss of uniqueness, which might be associated with bifurcation behavior, can only occur at states for which the constraint Jacobian is rank deficient or the system mass matrix fails to be positive definite for all kinematically admissible virtual velocities. Explicit implementation of the foregoing reduction to an independent set of differential equations, however, is not practical. Rather, computational algorithms that carry out this reduction numerically are presented in Section 7.3.

## 7.3 ALGORITHMS FOR SOLVING DIFFERENTIAL–ALGEBRAIC EQUATIONS

Efficient numerical integration algorithms are available for computing solutions of first-order systems of ordinary differential equations, with initial conditions given. Such methods calculate approximate solutions at grid points $t_i$ in time that are specified by the user, or under the control of a computer program, based on various forms of integration error control. One such class of methods is introduced in Section 7.4. Prior to delving into the details of numerical integration algorithms for first-order systems, however, it is instructive to first see that (1) second-order differential equations can be reduced to systems of first-order differential equations, and (2) mixed differential–algebraic equations can be treated by differential equation methods.

### 7.3.1 First-Order Initial-Value Problems

Consider first a second-order differential equation and the associated initial conditions, that is, a second-order initial-value problem,

$$\begin{aligned} \ddot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) \\ \mathbf{x}(t_0) &= \mathbf{x}^0 \\ \dot{\mathbf{x}}(t_0) &= \dot{\mathbf{x}}^0 \end{aligned} \tag{7.3.1}$$

By defining new variables $\mathbf{s} = \mathbf{x}$ and $\mathbf{r} = \dot{\mathbf{s}} = \dot{\mathbf{x}}$, a first-order initial-value problem that is equivalent to Eq. 7.3.1 may be written as

$$\begin{aligned} \dot{\mathbf{s}} &= \mathbf{r} \\ \dot{\mathbf{r}} &= \mathbf{f}(\mathbf{s}, \mathbf{r}, t) \\ \mathbf{s}(t_0) &= \mathbf{x}^0 \\ \mathbf{r}(t_0) &= \dot{\mathbf{x}}^0 \end{aligned} \tag{7.3.2}$$

This first-order initial-value problem may now be integrated for $\mathbf{s}$ and $\mathbf{r}$, and equivalently for $\mathbf{x}$ and $\dot{\mathbf{x}}$.

---

**Example 7.3.1:** The differential equation of motion for the simple pendulum derived in Example 7.2.1, with $\delta\phi_1 = v$, is

$$(J' + m)\ddot{\phi}_1 = -mg \cos \phi_1$$

Let initial conditions on $\phi_1$ be (see Fig. 6.3.1)

$$\phi_1(0) = \frac{3\pi}{2} \text{ rad}$$

$$\dot{\phi}_1(0) = 1 \text{ rad/s}$$

Defining $s \equiv \phi_1$ and $r \equiv \dot{s} = \dot{\phi}_1$, the second-order differential equation becomes

$$\dot{s} = r$$

$$\dot{r} = -\frac{mg}{(J' + m)} \cos s$$

and the initial conditions are

$$r(0) = \frac{3\pi}{2}$$

$$s(0) = 1$$

---

A vast literature is available that presents a relatively complete theory of the existence, uniqueness, and stability of solutions of first-order initial-value problems [44]. A single theorem is given here that provides valuable results, even with modest assumptions on the smoothness of the functions involved. For a more extensive treatment, the reader may consult references such as Reference 44.

Consider a vector variable $\mathbf{x}$ that depends on time and a continuously differentiable vector function $\mathbf{f}$ that depends on $\mathbf{x}$ and time; that is,

$$\mathbf{x}(t) = [x_1(t), \ldots, x_m(t)]^T$$

$$\mathbf{f}(\mathbf{x}, t) = [f_1(\mathbf{x}, t), \ldots, f_m(\mathbf{x}, \mathrm{t})]^T$$

A system of first-order, nonlinear, ordinary differential equations may be written in the general form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \qquad \mathbf{(7.3.3)}$$

This formulation includes the first-order differential equations of Eq. 7.3.2.

As is known [44], many solutions of such differential equations may exist. To obtain a unique solution, initial conditions may be specified at some time $t_0$, in the form

$$\mathbf{x}(t_0) = \mathbf{x}^0 \qquad \mathbf{(7.3.4)}$$

where $\mathbf{x}^0$ is a specified vector of initial values. The combination of the differential equation of Eq. 7.3.3 and initial conditions of Eq. 7.3.4 is an *initial-value problem* of ordinary differential equations. The existence and uniqueness theory for such initial-value problems, under rather weak hypotheses, is presented in the literature. An adequate result, for the purposes of the present study of dynamics, is the following [44]:

***Initial-Value Problem Existence Theorem:*** Let $\mathbf{f}$ be continuously differentiable in its arguments, for $|x_i| \le k$, $i = 1, \ldots, m$ and $t_0 \le t \le t_1$, and let $|x_i^0| < k$, $i = 1, \ldots, m$, where $k > 0$ is a constant. Then there exists a unique solution $\mathbf{x}(t)$ of Eqs. 7.3.3 and 7.3.4 in an interval $t_0 \le t < t^*$, such that $|x_i(t)| \le k$, $i = 1, \ldots, m$.

It is important to note that this theorem guarantees the existence of a *local* *solution* of the initial-value problem of Eqs. 7.3.3 and 7.3.4. The local nature of the solution is dictated by the fact that instabilities in solutions of differential equations may arise, leading to solutions that diverge to infinity. Globally valid existence theorems for nonlinear ordinary differential equations are not generally available. Nevertheless, this theorem yields valuable information and permits the use of a numerical integration method based on the methods outlined in Section 7.4. Providing the solution process continues to the desired terminal time, confidence in the validity of the solution is assured. Only if the integration process diverges must deeper theoretical analysis of the properties of the solution be investigated.

The reduction of second-order differential equations to first-order form is the easy part of solving differential–algebraic equations. Four basic approaches that reduce differential–algebraic equations to differential equations, for purposes of numerical integration, are presented in the following subsections. The last algorithm combines the better aspects of the first three to achieve both reliability and efficiency.

### 7.3.2 Generalized Coordinate Partitioning

It was demonstrated in Section 7.2 that, if the constraint Jacobian has full row rank, it is theoretically possible to reduce the system of differential–algebraic equations of Eqs. 7.2.1 through 7.2.3 to a set of second-order differential equations in independent generalized coordinates. The generalized coordinate partitioning algorithm presented here implements this method using an implicit numerical reduction in place of the explicit reduction derived in Section 7.2. Based on the knowledge that a theoretical reduction exists and has attractive mathematical properties, Eq. 7.2.1 is solved numerically for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$. The $\ddot{\mathbf{v}}$ components in $\ddot{\mathbf{q}}$ are simply extracted. They are identical to the values that would have been obtained if the reduction that leads to Eq. 7.2.14 had been carried out and the equation solved. An apparent disadvantage in this approach is that the dimension of the coefficient matrix in Eq. 7.2.1 is much greater than that of the coefficient matrix in Eq. 7.2.14. This disadvantage is, however, offset by the fact that (1) the coefficient matrix in Eq. 7.2.1 is sparse [13, 14, 46] and can be efficiently solved by well-developed sparse matrix computer software [47], and (2) the matrix inversions required in forming Eq. 7.2.14 can be avoided. The independent accelerations can then be integrated using the method of Section 7.4, and dependent variables can be obtained by solving the kinematic position and velocity equations.

A computational implementation of this approach, called the *generalized coordinate partitioning algorithm,* may be carried out as follows:

1. Begin with an assembled configuration and initial conditions that satisfy Eqs. 7.2.2 through 7.2.5 at the initial time $t_0$, that is, with $\mathbf{q}(t_0)$ and $\dot{\mathbf{q}}(t_0)$

that satisfy constraints. Such an initial configuration may be obtained using the methods presented in Section 4.3.

2. Evaluate and factor $\mathbf{\Phi_q}(t_0)$, as in Eq. 4.4.8, to determine dependent and independent variables $\mathbf{u}$ and $\mathbf{v}$, respectively. This partitioning of the generalized coordinate vector $\mathbf{q}$ will be retained until computational tests indicate that a new partitioning is required.
3. At a typical time step $t_i$, solve Eq. 7.2.1 for $\ddot{\mathbf{u}}$, $\ddot{\mathbf{v}}$, and $\boldsymbol{\lambda}$.
4. Integrate the first-order system

$$\begin{aligned}\dot{\mathbf{r}} &= \ddot{\mathbf{v}} \\ \dot{\mathbf{s}} &= \mathbf{r}\end{aligned}$$

from $t_i$ to $t_{i+1}$, using the integration algorithm of Section 7.4, to obtain $\mathbf{v}(t_{i+1})$ and $\dot{\mathbf{v}}(t_{i+1})$. Use the same integration algorithm and the first-order system

$$\begin{aligned}\dot{\mathbf{x}} &= \ddot{\mathbf{u}} \\ \dot{\mathbf{y}} &= \mathbf{x}\end{aligned}$$

to approximate $\mathbf{u}(t_{i+1})$ without error control (see Section 7.4).
5. Solve Eq. 7.2.2 for $\mathbf{u}(t_{i+1})$ using the Newton–Raphson method, starting with the approximation obtained in step (4). Solve Eq. 7.2.10 for $\dot{\mathbf{u}}(t_{i+1})$.
6. If $t_{i+1}$ exceeds the final time, stop. Otherwise, update $i$ to $i+1$ and continue.
7. If $\mathbf{\Phi_u}$ is ill-conditioned (see Section 4.4.1) or if the numerical integration algorithm requires multiple prediction iterations [31, 36], return to step (2) with $t_0$ replaced by $t_{i+1}$. Otherwise, return to step (3).

This implementation of the generalized coordinate partitioning algorithm has been used extensively [19] and has been found to be reliable and accurate. It satisfies constraints to the precision that is specified by the user and maintains good error control. It suffers from somewhat poorer numerical efficiency than alternative algorithms due to the requirement for iterative solution for dependent generalized coordinates $\mathbf{u}$ and dependent velocities $\dot{\mathbf{u}}$ in step (5).

### 7.3.3 Direct Integration

As observed in Section 7.2, Eq. 7.2.1 can be solved numerically for $\ddot{\mathbf{q}}$. If errors in satisfying constraints are ignored, the methods of Section 7.4 can be applied to integrate for $\mathbf{q}$ and $\dot{\mathbf{q}}$. This approach suffers from an accumulation of constraint error and may lead to substantial violation of the position and velocity constraint equations of Eqs. 7.2.2 and 7.2.3. For moderately regular system dynamics applications and small intervals of time, this approach may be satisfactory. It forms the basis for the *direct integration algorithm*.

1. Begin with an assembled configuration that satisfies Eqs. 7.2.2 through

7.2.5 at $t_0$, that is, with $\mathbf{q}(t_0)$ and $\dot{\mathbf{q}}(t_0)$, as in step (1) of the generalized coordinate partitioning algorithm.

2. At a typical time step $t_i$, solve Eq. 7.2.1 for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$.
3. Integrate the first-order system

$$\begin{aligned}\dot{\mathbf{r}} &= \ddot{\mathbf{q}}\\ \dot{\mathbf{s}} &= \mathbf{r}\end{aligned} \tag{7.3.5}$$

   from $t_i$ to $t_{i+1}$, using the algorithm of Section 7.4, to obtain $\mathbf{q}(t_{i+1})$ and $\dot{\mathbf{q}}(t_{i+1})$.
4. If $t_{i+1}$ exceeds the final time, terminate. Otherwise, update $i$ to $i+1$ and return to step (2).

The direct integration algorithm is simple, easy to implement, and computationally fast. It suffers, however, from a lack of error control on the constraints and may lead to erroneous results. Since no provision is made to control the accumulation of constraint error, the user must monitor constraint error and terminate the process if unacceptable constraint errors are encountered.

### 7.3.4 Constraint Stabilization

The second of Eqs. 7.2.1 is obtained by taking two time derivatives of the kinematic constraint equations of Eq. 7.2.2 and may be written in the form

$$\ddot{\boldsymbol{\Phi}} = \boldsymbol{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} - \boldsymbol{\gamma} = \mathbf{0} \tag{7.3.6}$$

From the control literature, it is known that numerical solution of the equation $\ddot{\boldsymbol{\Phi}} = \mathbf{0}$ can be unstable; that is, it can lead to values of $\boldsymbol{\Phi}$ and $\dot{\boldsymbol{\Phi}}$ that are far from $\mathbf{0}$. Baumgarte [40] observed, however, that the modified acceleration equation

$$\ddot{\boldsymbol{\Phi}} + 2\alpha\dot{\boldsymbol{\Phi}} + \beta^2\boldsymbol{\Phi} = \mathbf{0} \tag{7.3.7}$$

with $\alpha > 0$ and $\beta \neq 0$ is stable, hence implying that $\dot{\boldsymbol{\Phi}} \approx \boldsymbol{\Phi} \approx \mathbf{0}$. This observation forms the basis of the *constraint stabilization method* presented by Baumgarte [40], in which Eq. 7.3.7 is used in Eq. 7.2.1 instead of Eq. 7.3.6. To implement this method, Eq. 7.3.7 is written explicitly in the form

$$\boldsymbol{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} = \boldsymbol{\gamma} - 2\alpha(\boldsymbol{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \boldsymbol{\Phi}_t) - \beta^2\boldsymbol{\Phi} \equiv \hat{\boldsymbol{\gamma}} \tag{7.3.8}$$

where $\hat{\boldsymbol{\gamma}}$ replaces $\boldsymbol{\gamma}$ in Eq. 7.2.1.

With this replacement, the *constraint stabilization integration algorithm* consists simply of carrying out direct integration, as in Section 7.3.3. This approach has been demonstrated to be more stable and accurate than the elementary direct integration algorithm of Section 7.3.3 and is essentially as fast computationally. No general and uniformly valid method of selecting $\alpha$ and $\beta$, however, has been found. Furthermore, in the vicinity of kinematically singular configurations, even though Eq. 7.3.6 may have a well-behaved solution, the additional terms on the right side of Eq. 7.3.8 often lead to divergence of the algorithm.

## 7.3.5 A Hybrid Algorithm

The generalized coordinate partitioning and constraint stabilization algorithms are drastically different and have interesting complementary performance characteristics. A hybrid algorithm has been developed by Park [42] to take advantage of the better features of both methods. This algorithm follows the basic idea of generalized coordinate partitioning in that it defines a set of independent generalized coordinates, based on factorization of the constraint Jacobian $\mathbf{\Phi_q}$. It employs the modified acceleration equation of Eq. 7.3.8 during integration to stabilize small-amplitude oscillations in constraint error. Constraint errors are monitored, and the correction step in the generalized coordinate partitioning algorithm that solves the constraint equations for dependent coordinates $\mathbf{u}$ and $\dot{\mathbf{u}}$ is applied only when position and velocity constraint errors exceed a preset error tolerance. To avoid the instability that may occur with the constraint stabilization algorithm in the neighborhood of a kinematically singular configuration, upon occurrence of ill-conditioning in the coefficient matrix of Eq. 7.2.1, the algorithm reverts to pure generalized coordinate partitioning. Thus, the hybrid algorithm retains the reliability and positive error control characteristics of the generalized coordinate partitioning algorithm and approaches the computational speed that is achievable with the constraint stabilization algorithm.

The *hybrid numerical integration algorithm* is as follows:

1. Begin with an assembled configuration $\mathbf{q}(t_0)$ and $\dot{\mathbf{q}}(t_0)$ that satisfies Eqs. 7.2.2 through 7.2.5 at $t_0$, as in step (1) of the generalized coordinate partitioning algorithm of Section 7.3.1.
2. Evaluate and factor the matrix $\mathbf{\Phi_q}(t_0)$, as in Eq. 4.4.8, to determine dependent and independent variables $\mathbf{u}$ and $\mathbf{v}$, respectively. This partitioning of the generalized coordinate vector $\mathbf{q}$ will be retained until computational tests indicate the need for a new partitioning.
3. Solve Eq. 7.2.1 for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$ without stabilization terms.
4. Integrate the first-order system

$$\dot{\mathbf{r}} = \ddot{\mathbf{q}}$$
$$\dot{\mathbf{s}} = \mathbf{r}$$

   from $t_i$ to $t_{i+1}$, as in the algorithm of Section 7.3.2, to obtain $\mathbf{q}(t_{i+1})$ and $\dot{\mathbf{q}}(t_{i+1})$, with error control only on $\mathbf{v}$ and $\dot{\mathbf{v}}$.
5. **a.** If $\mathbf{\Phi}(\mathbf{q}, t)$ at $t_{i+1}$ is within the specified constraint error tolerance, accept the integrated values of $\mathbf{q}(t_{i+1})$ and $\dot{\mathbf{q}}(t_{i+1})$. To compensate for error that may accumulate in continued integration, constraint stabilization terms are used until an unacceptable level of error is encountered. To evaluate the correction terms, multiply $\beta^2$ by the

previously evaluated constraint violation. For the velocity violation term, evaluate the Jacobian and $\mathbf{\Phi}_t$ and calculate $2\alpha(\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{\Phi}_t)$. Add these constraint and velocity equation violation terms to $\boldsymbol{\gamma}$ to form $\hat{\boldsymbol{\gamma}}$, and solve Eq. 7.2.1, with $\boldsymbol{\gamma}$ replaced by $\hat{\boldsymbol{\gamma}}$, for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$.

**b.** If the integrated values of $\mathbf{q}(t_{i+1})$ do not satisfy the constraint error tolerance, use the Newton–Raphson method to iteratively solve Eq. 7.2.2 for accurate values of $\mathbf{u}(t_{i+1})$. After position analysis, solve Eq. 7.2.3 for $\dot{\mathbf{u}}(t_{i+1})$. Since position and velocity are corrected using the constraint equations, use Eq. 7.2.1 to solve for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$. Evaluate matrix conditioning and integration performance to check if the independent variable set is still valid [45]. If the independent variable set is to be changed, set a flag so that after the correction step integration will be restarted with the new independent variables from step (2).

6. Provide velocity and acceleration to the integration algorithm and use independent variable error estimates to determine the integration time step and order (see Section 7.4).
7. If $t_{i+1}$ exceeds the final time, terminate. Otherwise, update $t_{i+1}$ and return to step (4).

This method will perform best with an optimized choice of $\alpha$ and $\beta$ at each time step, but to date there is no general method of selecting $\alpha$ and $\beta$. The algorithm uses small fixed values of $\alpha$ and $\beta$ to damp out a moderate amount of constraint error, to prevent adversely affecting the integration algorithm by the correction term. This algorithm is the most reliable of those discussed in this section. It is employed in the DADS computer code [27].

## 7.4 NUMERICAL INTEGRATION OF FIRST-ORDER INITIAL-VALUE PROBLEMS

Algorithms for solving differential–algebraic equations of machine dynamics presented in Section 7.3 provide alternative methods of reducing differential–algebraic equations to initial-value problems that can be integrated using standard numerical solution techniques. The purpose of this section is to present basic ideas and algorithms that may be used for this purpose. Regardless of the algorithm selected in Section 7.3, the task remains to integrate an initial-value problem of the form

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, t)\\ \mathbf{x}(t_0) &= \mathbf{x}^0\end{aligned} \tag{7.4.1}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$ is the vector of variables to be integrated and the function $\mathbf{f}(\mathbf{x}, t)$ on the right of Eq. 7.4.1 is defined by the computational sequence of the algorithms employed in Section 7.3. The constrained dynamic existence theorem of Section 6.3.3 and the implicit function theorem guarantee that, if the constraint Jacobian has full row rank and if its entries and the applied generalized force are continuously differentiable, then the hypotheses of the initial-value problem existence theorem are satisfied for each of the algorithms. The vector $\mathbf{x}^0$ defines the initial conditions.

Equation 7.4.1 comprises an initial-value problem that is to be integrated numerically. Methods of polynomial interpolation, upon which most numerical integration algorithms are based, are presented in this section and used to derive and demonstrate properties of numerical integration algorithms. Error estimates are presented to give an indication of the methods used for error control during the integration process. For a more detailed survey of numerical integration methods in dynamics, the reader may consult the survey of Reference 38.

## 7.4.1 Polynomial Interpolation

The fundamental idea used in numerical integration is the approximation of a function $f(t)$ by a polynomial $P(t)$, that is, a polynomial that agrees with $f$ and perhaps some of its derivatives at one or more points $t_i$. There are numerous methods by which approximating polynomials can be derived, each with its inherent advantages and disadvantages.

**Taylor Expansion** The Taylor expansion method is based on approximating a function $f(t)$ near a point $t_0$ by the *Taylor polynomial* [25, 26]

$$T_k(t) = f(t_0) + f^{(1)}(t_0)(t-t_0) + \frac{f^{(2)}(t_0)(t-t_0)^2}{2!} + \cdots + \frac{f^{(k-1)}(t_0)(t-t_0)^{k-1}}{(k-1)!} \tag{7.4.2}$$

where $f^{(i)}(t_0)$ denotes the $i$th derivative of $f(t)$ evaluated at $t_0$. This polynomial agrees with $f$ and its first $k-1$ derivatives at $t_0$ (Prob. 7.4.1); that is,

$$T_k^{(j)}(t_0) = f^{(j)}(t_0), \qquad j = 0, 1, \ldots, k-1 \tag{7.4.3}$$

---

**Example 7.4.1:** Consider the use of Eq. 7.4.2 to approximate $e^t$ by expanding $f(t) = e^t$ about $t_0 = 0$. Since $f^{(j)} = e^t$, for all $j$, Eq. 7.4.2 becomes

$$\begin{aligned} T_k(t) &= e^0 + e^0(t-0) + \frac{e^0(t-0)^2}{2!} + \cdots + \frac{e^0(t-0)^{k-1}}{(k-1)!} \\ &= 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots + \frac{t^{(k-1)}}{(k-1)!} \end{aligned}$$

If $t = 1$, then $f(1) = e$ is approximated by

$$e \approx T_k(1) = 1 + 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \cdots + \frac{1^{(k-1)}}{(k-1)!}$$

The value of $e$ to ten-place accuracy is $e = 2.718281828$ so $e^{1/2} = 1.648721271$. Table 7.4.1 lists $T_k(t)$ and the error $E_k(t) = T_k(t) - e^t$ for various values of $k$ and $t = \frac{1}{2}$ and $t = 1$.

**TABLE 7.4.1 Taylor Approximate Values of $e^t$**

| $k$ | $T_k(0.5)$ | $E_k(0.5)$ | $T_k(1)$ | $E_k(1)$ |
|---|---|---|---|---|
| 1 | 1.000000000 | −0.6487212707 | 1.000000000 | −1.7182818285 |
| 2 | 1.500000000 | −0.1487212707 | 2.000000000 | −0.7182818285 |
| 3 | 1.625000000 | −0.0237212707 | 2.500000000 | −0.2182818285 |
| 4 | 1.645833333 | −0.0028879374 | 2.666666667 | −0.0516151618 |
| 5 | 1.648437500 | −0.0002837707 | 2.708333333 | −0.0099484951 |
| 6 | 1.648697917 | −0.0000233540 | 2.716666667 | −0.0016151618 |
| 7 | 1.648719618 | −0.0000016526 | 2.718055556 | −0.0002262729 |
| 8 | 1.648721168 | −0.0000001025 | 2.718253968 | −0.0000278602 |
| 9 | 1.648721265 | −0.0000000057 | 2.718278770 | −0.0000030586 |
| 10 | 1.648721270 | −0.0000000003 | 2.718281526 | −0.0000003029 |
| 11 | 1.648721271 | 0.0000000000 | 2.718281801 | −0.0000000273 |
| 12 | 1.648721271 | 0.0000000000 | 2.718281826 | −0.0000000023 |
| 13 | 1.648721271 | 0.0000000000 | 2.718281828 | −0.0000000002 |
| 14 | 1.648721271 | 0.0000000000 | 2.718281828 | 0.0000000000 |

From this example it is clear that a Taylor polynomial approximation may be used to evaluate a function at nearby points to any desired accuracy. It also illustrates that accuracy of the Taylor approximation is improved when $|t - t_0|$ is reduced and when more terms are included in the expansion of Eq. 7.4.2, that is, when higher order derivatives are used.

In general, the values of a function $f(t)$ at several values of $t$ are easier to obtain than high-order derivatives of $f$ at a single value of $t$, so Taylor expansion is not well suited for general-purpose integration algorithms. It is, however, useful in understanding how other interpolating polynomials are developed into integration algorithms and in deriving numerical integration error estimates.

**Interpolating Polynomials** A polynomial $P_k(t)$ of degree $k-1$ can be constructed that agrees with $f(t)$ at $k$ distinct points $t_{n-k+1}, \ldots, t_{n-1}, t_n$; that is,

$$P_k(t_i) = f(t_i), \qquad i = n-k+1, \ldots, n-1, n$$

as shown in Fig. 7.4.1. There is one and only one polynomial of degree $k-1$ that satisfies these interpolating conditions [31]. With equally spaced points, $t_i - t_{i-1} = h$, this polynomial is constructed in the form of a *Newton backward difference polynomial* [31]

$$P_k(t) = f_n + \sum_{i=1}^{k-1} \frac{\nabla^i f_n}{(i)!\, h^i} \prod_{j=0}^{i-1} (t - t_{n-j}) \tag{7.4.4}$$

where the *product notation* (analogous to $\sum_{j=0}^{\ell}$ summation notation)

$$\prod_{j=0}^{\ell} g_j \equiv g_0 \times g_1 \times \cdots \times g_{\ell-1} \times g_\ell$$

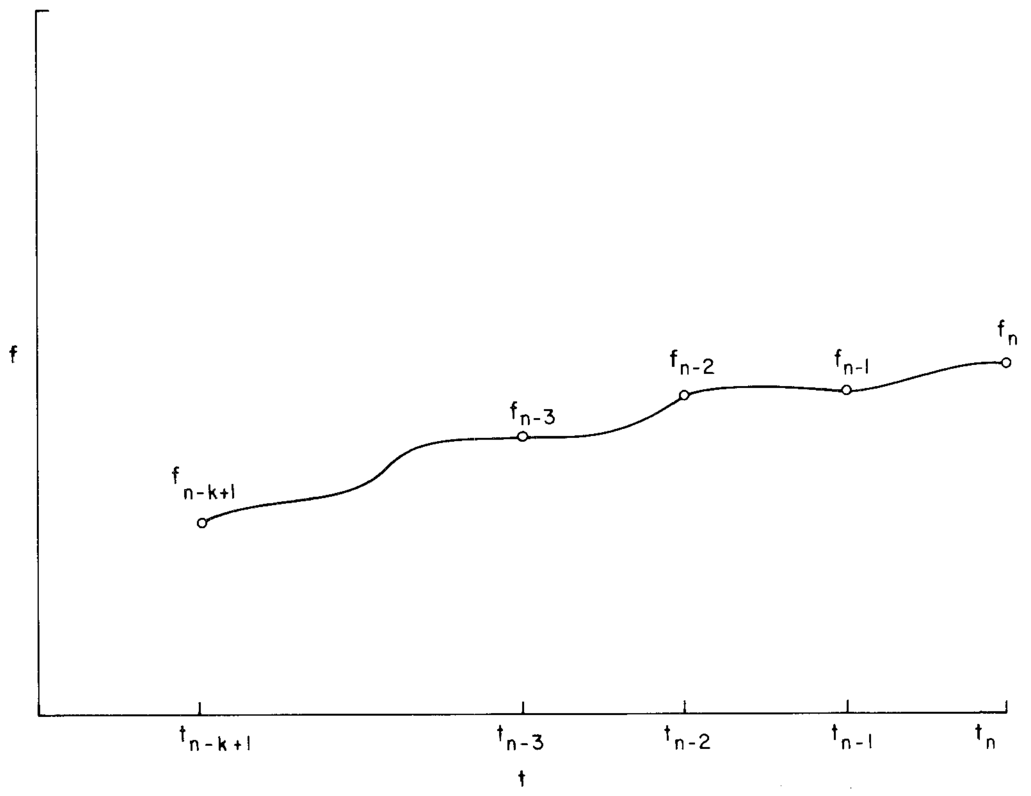

**Figure 7.4.1** Interpolation through $k$ points.

is used and the *backward differences* $\nabla^i f_n$ are defined as

$$\begin{aligned} &f_i = f(t_i), \qquad i = n-k+1, \ldots, n \\ &\nabla^1 f_n = f_n - f_{n-1} \\ &\nabla^2 f_n = \nabla^1 f_n - \nabla^1 f_{n-1} \\ &\vdots \\ &\nabla^{k-1} f_n = \nabla^{k-2} f_n - \nabla^{k-2} f_{n-1} \end{aligned}$$

The reader is encouraged to verify that Eq. 7.4.4 provides the desired values $f_i$ for $k = 1, 2,$ and 3 (Prob. 7.4.2).

**Example 7.4.2:** Consider the use of Eq. 7.4.4 to approximate $e^t$ near $t = 1$ using Newton backward difference polynomials. Let $t_n = 1$ and $h = 0.1$. $P_k(1.1)$ and $P_k(1.2)$ are evaluated for $k = 1, 2, \ldots, 10$ and tabulated in Table 7.4.2. The error $E_k(t) = P_k(t) - e^t$ is also tabulated. As expected, the higher the order of the polynomial and the smaller $|t - 1|$, the better the approximation.

TABLE 7.4.2 Newton Backward Difference Approximation of $e^t$

| $k$ | $P_k(1.1)$ | $E_k(1.1)$ | $P_k(1.2)$ | $E_k(1.2)$ |
|---|---|---|---|---|
| 1 | 2.718281828 | −0.285884195 | 2.718281828 | −0.601835094 |
| 2 | 2.976960546 | −0.027205478 | 3.235639263 | −0.084477660 |
| 3 | 3.001577080 | −0.002588944 | 3.309488867 | −0.010628056 |
| 4 | 3.003919653 | −0.000246371 | 3.318859159 | −0.001257764 |
| 5 | 3.004142579 | −0.000023445 | 3.319973785 | −0.000143137 |
| 6 | 3.004163793 | −0.000002231 | 3.320101070 | −0.000015852 |
| 7 | 3.004165812 | −0.000000212 | 3.320115202 | −0.000001721 |
| 8 | 3.004166004 | −0.000000020 | 3.320116739 | −0.000000184 |
| 9 | 3.004166022 | −0.000000002 | 3.320116903 | −0.000000019 |
| 10 | 3.004166024 | 0.000000000 | 3.320116921 | −0.000000002 |
| 11 | 3.004166024 | 0.000000000 | 3.320116923 | 0.000000000 |

**Interpolation Error** In solving differential equations, an interpolation polynomial on one set of data points needs to be transformed to an interpolation polynomial on another set that is obtained by dropping the left point and adding a new point on the right, to march ahead on a time grid. The form of Taylor and Newton backward difference polynomials suggests that it may be possible to take advantage of the fact that most of the data remain the same on the two grids. To change from approximating $f$ near $t_0$ using $f_0, \dots, f_0^{(k-1)}$ to using $f_0, \dots, f_0^{(k)}$, by Taylor polynomials, Eq. 7.4.2 yields

$$T_{k+1}(t) = T_k(t) + \frac{f_0^{(k)}}{(k)!}(t - t_0)^k$$

To achieve something comparable in the case of increasing the data set from $k$ to $k+1$ points for Newton backward difference interpolation requires that

$$P_{k+1}(t) = P_k(t) + R_{k+1}(t) \tag{7.4.5}$$

where $R_{k+1}(t)$ must be of degree $k$. That is, adding a data point amounts to adding a correction term to the current interpolating polynomial that is of higher degree. Substituting Eq. 7.4.4 into Eq. 7.4.5 and solving for $R_{k+1}(t)$,

$$R_{k+1}(t) = \frac{\nabla^k f_n}{(k)!\,h^k} \prod_{j=0}^{k-1} (t - t_{n-j}) \tag{7.4.6}$$

It is shown in References 31, 36, and 37 that

$$\frac{\nabla^k f_k}{(k)!\,h^k} = \frac{f^{(k)}(\xi)}{(k)!} \tag{7.4.7}$$

for some $\xi$ in $[t_{n-k}, t_n]$. The error due to interpolating $f(t)$ by $P_k(t)$ is estimated, assuming $f(t) \approx P_{k+1}(t)$, by combining Eqs. 7.4.6 and 7.4.7,

$$f(t) - P_k(t) \approx R_{k+1}(t) = \frac{f^{(k)}(\xi)}{(k)!} \prod_{j=0}^{k-1} (t - t_{n-j}) \tag{7.4.8}$$

for some $\xi$ in $[t_{n-k}, t_n]$. How good the estimate is depends on how much $f^{(k)}$ varies over the interval $[t_{n-k}, t_n]$.

An important application of Newton backward difference polynomials is the use of past function values to extrapolate ahead in time, as was illustrated in Example 7.4.2. Equation 7.4.8 indicates that, if the $k$th derivative of the function being approximated grows rapidly, then approximation error increases. Even though the $k$th derivative of the function is not known, the error estimate on the right of Eq. 7.4.8 may be approximated by the backward difference term on the left of Eq. 7.4.7, which can be monitored during computation. If the error estimate indicates a problem, the time step $h$ may be reduced to decrease the magnitude of error.

**Use of Polynomial Interpolation in Numerical Integration** Interpolation methods for approximating the solution of Eq. 7.4.1., or

$$\begin{aligned}\dot{\mathbf{x}} &\equiv \mathbf{x}^{(1)}(t) = \mathbf{f}(\mathbf{x}, t)\\ \mathbf{x}(t_0) &= \mathbf{x}^0\end{aligned}$$

on an interval $[a, b]$ employ a mesh of grid points $\{t_0, t_1, \ldots\}$ in $[a, b]$, with $t_0 = a$. It is presumed that these grid points are equally spaced, in which case

$$t_n = t_0 + nh, \qquad n = 0, 1, \ldots$$

where $h$ is the time step. The notation $\mathbf{x}_n$ means an approximation of the solution $\mathbf{x}(t)$ of Eq. 7.4.1 at grid point $t_n$; that is,

$$\mathbf{x}_n \approx \mathbf{x}(t_n)$$

Because $\mathbf{x}(t)$ satisfies the differential equation, an approximation of $\mathbf{x}(t_n)$ leads to an approximation of $\mathbf{x}_n^{(1)}(t_n)$; that is,

$$\mathbf{f}_n = \mathbf{f}(\mathbf{x}_n, t_n) \approx \mathbf{x}^{(1)}(t_n) = \mathbf{f}(\mathbf{x}(t_n), t_n)$$

The basic computational task in numerical integration is to advance the numerical solution from $t_n$ to $t_{n+1}$, after having computed $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$. The exact solution can be obtained by integrating both sides of the differential equation from $t_n$ to $t_{n+1}$ to obtain

$$\begin{aligned}\mathbf{x}(t_{n+1}) &= \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{x}^{(1)}(t)\, dt\\ &= \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{x}(t), t)\, dt \qquad \textbf{(7.4.9)}\end{aligned}$$

The *Adams family of numerical integration methods* approximate this solution by replacing $\mathbf{f}(x(\mathbf{t}), t)$ in the interval $[t_n, t_{n+1}]$ by a polynomial that interpolates past computed values $\mathbf{f}_i$, which is then integrated.

**Example 7.4.3:** The simplest application of polynomial interpolation for numerical integration is to interpolate $\mathbf{f}(\mathbf{x}(t), t)$ by Eq. 7.4.4 with $k = 1$; that is, in $[t_n, t_{n+1}]$,

$$\mathbf{f}(\mathbf{x}(t), t) \approx \mathbf{f}_n \equiv \mathbf{f}(\mathbf{x}_n, t_n)$$

Using this relation in Eq. 7.4.9,

$$\begin{aligned} \mathbf{x}_{n+1} \approx \mathbf{x}(t_{n+1}) &\approx \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{f}_n \, dt \\ &= \mathbf{x}(t_n) + \mathbf{f}_n(t_{n+1} - t_n) \\ &\approx \mathbf{x}_n + h\mathbf{f}_n \end{aligned}$$

where $t_{n+1} - t_n \equiv h$. This elementary approximation may be applied to the initial-value problem

$$\dot{x} = x$$
$$x(0) = 1$$

which has the unique solution $x(t) = e^t$. Approximate solutions are obtained with $h_1 = 0.1$ and $h_2 = 0.01$, with results tabulated at $t_i = 0.1i$ in Table 7.4.3.

**TABLE 7.4.3 Numerical Approximation of $e^t$ with $k = 1$**

| $i$ | $t_i$ | $e^{t_i}$ | $x_i$ ($h = 0.1$) (% error) | $x_{10i}$ ($h = 0.01$) (% error) |
|---|---|---|---|---|
| 0 | 0.0 | 1.0000000000 | 1.0000000000 (0.000) | 1.0000000000 (0.000) |
| 1 | 0.1 | 1.1051709181 | 1.1000000000 (0.468) | 1.1046221254 (0.050) |
| 2 | 0.2 | 1.2214027582 | 1.2100000000 (0.934) | 1.2201900399 (0.099) |
| 3 | 0.3 | 1.3498588076 | 1.3310000000 (1.397) | 1.3478489153 (0.149) |
| 4 | 0.4 | 1.4918246976 | 1.4641000000 (1.858) | 1.4888637336 (0.198) |
| 5 | 0.5 | 1.6487212707 | 1.6105100000 (2.318) | 1.6446318218 (0.248) |
| 6 | 0.6 | 1.8221188004 | 1.7715610000 (2.775) | 1.8166966986 (0.298) |
| 7 | 0.7 | 2.0137527075 | 1.9487171000 (3.230) | 2.0067633684 (0.347) |
| 8 | 0.8 | 2.2255409285 | 2.1435888100 (3.682) | 2.2167152172 (0.397) |
| 9 | 0.9 | 2.4596031112 | 2.3579476910 (4.133) | 2.4486326746 (0.446) |
| 10 | 1.0 | 2.7182818285 | 2.5937424601 (4.582) | 2.7048138294 (0.495) |

As expected, the accuracy of the numerical results is much better with the smaller step size. It must be recalled, however, that 100 time steps are required to reach $t = 1.0$, with the step size $h = 0.01$. Roughly speaking, you pay for the accuracy that you get.

### 7.4.2 Adams–Bashforth Predictor

The Adams–Bashforth method of approximating the solution of the initial-value problem of Eq. 7.4.1 is based on approximating the function $\mathbf{f}(\mathbf{x}(t), t)$ on the right of Eq. 7.4.9 by the Newton backward difference polynomial of Eq. 7.4.4. This presumes that $\mathbf{f}_{n-k+1}, \ldots, \mathbf{f}_n$ are available as approximations of $\mathbf{f}(\mathbf{x}(t), t)$ at $t_{n-k+1}, \ldots, t_n$, with $t_i - t_{i-1} = h$. The algorithm that is used to construct these approximations will become apparent soon. For now, presume they are known. Substituting from Eq. 7.4.4 for $\mathbf{f}(\mathbf{x}(t), t)$ into Eq. 7.4.9 and integrating the polynomial of degree $k-1$ yields a polynomial of degree $k$ in $t$, which is evaluated at $t = t_{n+1}$. The result is called the *Adams–Bashforth formula of order k.*

A convenient notation for the polynomial of degree $k-1$ that is employed in the Adams–Bashforth formula of order $k$ at $t_n$, which interpolates the computed values of $\mathbf{f}_i$ at $k$ preceding points, is

$$\mathbf{P}_{k,n}(t_{n+1-j}) = \mathbf{f}_{n+1-j}, \qquad j = 1, 2, \ldots, k$$

An approximation, or prediction, of the solution at $t_{n+1}$ is obtained as

$$\mathbf{x}^p_{n+1} = \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{P}_{k,n}(t)\, dt \tag{7.4.10}$$

Newton's backward difference polynomial from Eq. 7.4.4 is

$$\mathbf{P}_{k,n}(t) = \mathbf{f}_n + \sum_{i=1}^{k-1} \frac{\nabla^i \mathbf{f}_n}{(i)!\, h^i} \prod_{j=0}^{i-1} (t - t_{n-j})$$

This may be substituted into Eq. 7.4.10 to obtain the *Adams–Bashforth predictor of order k*:

$$\mathbf{x}^p_{n+1} = \mathbf{x}_n + h \sum_{i=1}^{k} \gamma_{i-1} \nabla^{i-1} \mathbf{f}_n \tag{7.4.11}$$

where $\nabla^0 \mathbf{f}_n = \mathbf{f}_n$ and

$$\begin{aligned} \gamma_0 &= 1 \\ \gamma_i &= \frac{1}{(i)!\, h} \int_{t_n}^{t_{n+1}} \prod_{j=0}^{i-1} \frac{t - t_{n-j}}{h}\, dt, \qquad i = 1, 2, \ldots \end{aligned} \tag{7.4.12}$$

which are called *Adams–Bashforth coefficients.*

To see that the values of $\gamma_i$ are independent of $h$, introduce the change of

**TABLE 7.4.4 Adams–Bashforth Coefficients**

| $\gamma_0$ | $\gamma_1$ | $\gamma_2$ | $\gamma_3$ | $\gamma_4$ | $\gamma_5$ |
|---|---|---|---|---|---|
| 1 | $\frac{1}{2}$ | $\frac{5}{12}$ | $\frac{3}{8}$ | $\frac{251}{720}$ | $\frac{95}{298}$ |

variable $s = (t - t_n)/h$ in Eq. 7.4.12 to obtain (Prob. 7.4.3)

$$\gamma_i = \frac{1}{(i)!}\int_0^1 \prod_{j=0}^{i-1} (s + j)\, ds$$

since $(t - t_{n-j})/h = [t - t_n + (t_n - t_{n-j})]/h = s + j$. Numerical values of the first six Adams–Bashforth coefficients are given in Table 7.4.4.

The Adams–Bashforth formula of Eq. 7.4.11 is called an *explicit method,* because the term $x_{n+1}$ appears only on the left side of the equation and is determined explicitly by prior data. Note that at $t_0$ there are no past data points to interpolate, so the order $k$ must be selected as 1. Thus, from Eq. 7.4.11 with $k = 1$,

$$\mathbf{x}_1 = \mathbf{x}_0 + h\mathbf{f}_0$$

At $t_1$, there is now one past data point, so the order $k = 2$ can be selected, and, from Eq. 7.4.11 and Table 7.4.4,

$$\begin{aligned}\mathbf{x}_2 &= \mathbf{x}_1 + h(\mathbf{f}_1 + \mathbf{f}_1 - \mathbf{f}_0)\\ &= \mathbf{x}_1 + h(2\mathbf{f}_1 - \mathbf{f}_0)\end{aligned}$$

This process can be repeated to increase the order $k$ of the Adams–Bashforth predictor as more past integration points $t_i$, where $\mathbf{f}_i$ is known, are available. This process of increasing order $k$, from 1 up to the desired order, is called *starting the Adams–Bashforth algorithm.*

The approximation $\mathbf{x}^p_{n+1}$ in Eq. 7.4.11 has two sources of error. One source is due to approximating $\mathbf{x}^{(1)}(t)$ by an interpolating polynomial, known as *local truncation error.* The other source of error is errors that are present in the previously calculated values $\mathbf{x}_n, \mathbf{x}_{n-1}, \ldots$ . It has been rigorously shown [31, 36, 37] that the latter contribution is insignificant when compared to local truncation error.

It is shown in Reference 31 that the local truncation error is

$$\boldsymbol{\tau}^p_{n+1} = \gamma_k h^{k+1}\mathbf{x}^{(k+1)}(\xi) \approx h\gamma_k \nabla^k \mathbf{f}_n \tag{7.4.13}$$

for constant step size $h$ and some $\xi$ in $[t_{n+1-k}, t_{n+1}]$. Note that if enough differences are kept and if $h < 1$ it is possible to adjust the order and/or to decrease the step size to make the local truncation error less than some desired error tolerance.

**Example 7.4.4:** Consider again the initial-value problem of Example 7.4.3:

$$\dot{x} = x$$
$$x(0) = 1$$

The solution is $x(t) = e^t$. A numerical approximation of the solution is to be constructed using Eq. 7.4.11 with $h = 0.1$. Backward differences calculated at $t_i$, using associated $f_i$, are shown in Table 7.4.5. Notice that at $t = 0$ the only available information is $x(0) = 1$. Hence, $k$ must be 1 at $t = 0$. From Eq. 7.4.11,

$$\begin{aligned} x_1 &= x_0 + h(\gamma_0 \nabla^0 f_0) \\ &= 1 + (0.1)(1)(1) = 1.1 \end{aligned}$$

At $t = 0.1$, $x_1$ and $x_0$ are available. Hence, $\nabla^1 f_1$ can be evaluated, as shown in Table 7.4.5. Therefore, from Eq. 7.4.11 with $k = 2$,

$$\begin{aligned} x_2 &= x_1 + h(\gamma_0 \nabla^0 f_1 + \gamma_1 \nabla^1 f_1) \\ &= 1.1 + (0.1)\{(1)(1.1) + (\tfrac{1}{2})(0.1)\} \\ &= 1.215 \end{aligned}$$

Moving to $t_3, t_4, \ldots$, using successively larger values of order $k$ in Eq. 7.4.11,

$$\begin{aligned} x_3 &= x_2 + h(\gamma_0 \nabla^0 f_2 + \gamma_1 \nabla^1 f_2 + \gamma_2 \nabla^2 f_2) \\ &= 1.215 + (0.1)\{(1)(1.215) + (\tfrac{1}{2})(0.115) + (\tfrac{5}{12})(0.015)\} \\ &= 1.342875 \end{aligned}$$

$$\begin{aligned} x_4 &= x_3 + h(\gamma_0 \nabla^0 f_3 + \gamma_1 \nabla^1 f_3 + \gamma_2 \nabla^2 f_3 + \gamma_3 \nabla^3 f_3) \\ &= 1.342875 + (0.1)\{(1)(1.342875) + (\tfrac{1}{2})(0.127875) \\ &\quad + (\tfrac{5}{12})(0.012875) + (\tfrac{3}{8})(-0.002125)\} \\ &= 1.484013 \end{aligned}$$

**TABLE 7.4.5 Newton Backward Differences**

| $k$ | $\nabla^i f$ | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
|---|---|---|---|---|---|---|
| 1 | $\nabla^0 f$ | 1.0 | | | | |
| 2 | $\nabla^0 f$ | 1.0 | 1.1 | | | |
| | $\nabla^1 f$ | | 0.1 | | | |
| 3 | $\nabla^0 f$ | 1.0 | 1.1 | 1.215 | | |
| | $\nabla^1 f$ | | 0.1 | 0.115 | | |
| | $\nabla^2 f$ | | | 0.015 | | |
| 4 | $\nabla^0 f$ | 1.0 | 1.1 | 1.215 | 1.342875 | |
| | $\nabla^1 f$ | | 0.1 | 0.115 | 0.127875 | |
| | $\nabla^2 f$ | | | 0.015 | 0.012875 | |
| | $\nabla^3 f$ | | | | −0.002125 | |
| 5 | $\nabla^0 f$ | 1.0 | 1.1 | 1.215 | 1.342875 | 1.484013 |
| | $\nabla^1 f$ | | 0.1 | 0.115 | 0.127875 | 0.141138 |
| | $\nabla^2 f$ | | | 0.015 | 0.012875 | 0.132630 |
| | $\nabla^3 f$ | | | | −0.002125 | 0.000385 |
| | $\nabla^4 f$ | | | | | 0.002510 |

TABLE 7.4.6 Adams–Bashforth Predictor Values of $e^{t_i}$, $h = 0.1$

| | | | $x_i$ (% error) | | | | |
|---|---|---|---|---|---|---|---|
| $i$ | $t_i$ | $e^{t_i}$ | $k=1$ | $k=2$ | $k=3$ | $k=4$ | $k=5$ |
| 0 | 0.0 | 1.0 | | | | | |
| 1 | 0.1 | 1.105171 | 1.100000 | 1.100000 | 1.100000 | 1.100000 | 1.100000 |
| | | | (0.468) | (0.468) | (0.468) | (0.468) | (0.468) |
| 2 | 0.2 | 1.221403 | 1.210000 | 1.215000 | 1.215000 | 1.215000 | 1.215000 |
| | | | (0.934) | (0.524) | (0.524) | (0.524) | (0.524) |
| 3 | 0.3 | 1.349859 | 1.331000 | 1.342250 | 1.342875 | 1.342875 | 1.342875 |
| | | | (1.397) | (0.564) | (0.517) | (0.517) | (0.517) |
| 4 | 0.4 | 1.491825 | 1.464100 | 1.482838 | 1.484093 | 1.484013 | 1.484013 |
| | | | (1.858) | (0.602) | (0.518) | (0.524) | (0.524) |
| 5 | 0.5 | 1.648721 | 1.610510 | 1.638151 | 1.640119 | 1.640038 | 1.640126 |
| | | | (2.318) | (0.641) | (0.522) | (0.527) | (0.521) |
| 6 | 0.6 | 1.822119 | 1.771561 | 1.809731 | 1.812549 | 1.812525 | 1.812679 |
| | | | (2.775) | (0.680) | (0.525) | (0.527) | (0.518) |
| 7 | 0.7 | 2.013753 | 1.948717 | 1.999284 | 2.003109 | 2.003146 | 2.003305 |
| | | | (3.230) | (0.719) | (0.529) | (0.527) | (0.519) |
| 8 | 0.8 | 2.225541 | 2.143589 | 2.208689 | 2.213703 | 2.213811 | 2.213988 |
| | | | (3.682) | (0.757) | (0.532) | (0.527) | (0.519) |
| 9 | 0.9 | 2.459603 | 2.357948 | 2.440029 | 2.446348 | 2.446631 | 2.446842 |
| | | | (4.133) | (0.796) | (0.535) | (0.527) | (0.519) |
| 10 | 1.0 | 2.718282 | 2.593742 | 2.695599 | 2.703641 | 2.703938 | 2.704177 |
| | | | (4.582) | (0.834) | (0.539) | (0.528) | (0.519) |

TABLE 7.4.7 Adams–Bashforth Predictor Values of $e^{t_i}$, $h = 0.01$

| | | | $x_i$ (% error) | | | | |
|---|---|---|---|---|---|---|---|
| $i$ | $t_i$ | $e^{t_i}$ | $k=1$ | $k=2$ | $k=3$ | $k=4$ | $k=5$ |
| 0 | 0.0 | 1.0 | | | | | |
| 1 | 0.0 | 1.010050 | 1.010000 | 1.010000 | 1.010000 | 1.010000 | 1.010000 |
| | | | (0.005) | (0.005) | (0.005) | (0.005) | (0.005) |
| 2 | 0.0 | 1.020201 | 1.020100 | 1.020150 | 1.020150 | 1.020150 | 1.020150 |
| | | | (0.010) | (0.005) | (0.005) | (0.005) | (0.005) |
| 3 | 0.0 | 1.030455 | 1.030301 | 1.030402 | 1.030403 | 1.030403 | 1.030403 |
| | | | (0.015) | (0.005) | (0.005) | (0.005) | (0.005) |
| 4 | 0.0 | 1.040811 | 1.040604 | 1.040758 | 1.040759 | 1.040748 | 1.040748 |
| | | | (0.020) | (0.005) | (0.005) | (0.005) | (0.005) |
| 5 | 0.1 | 1.051271 | 1.051010 | 1.051217 | 1.051218 | 1.051218 | 1.051218 |
| | | | (0.025) | (0.005) | (0.005) | (0.005) | (0.005) |
| 6 | 0.1 | 1.061837 | 1.061520 | 1.061781 | 1.061783 | 1.061783 | 1.061783 |
| | | | (0.030) | (0.005) | (0.005) | (0.005) | (0.005) |
| 7 | 0.1 | 1.072508 | 1.072135 | 1.072452 | 1.072454 | 1.072454 | 1.072454 |
| | | | (0.035) | (0.005) | (0.005) | (0.005) | (0.005) |
| 8 | 0.1 | 1.083287 | 1.082857 | 1.083230 | 1.083233 | 1.083233 | 1.083233 |
| | | | (0.040) | (0.005) | (0.005) | (0.005) | (0.005) |
| 9 | 0.1 | 1.094174 | 1.093685 | 1.094116 | 1.094119 | 1.094119 | 1.094119 |
| | | | (0.045) | (0.005) | (0.005) | (0.005) | (0.005) |
| 10 | 0.1 | 1.105171 | 1.104622 | 1.105112 | 1.105115 | 1.105115 | 1.105115 |
| | | | (0.050) | (0.005) | (0.005) | (0.005) | (0.005) |

The foregoing process of implementing Eq. 7.4.11, starting with $k = 1$ and increasing the order to some desired value of $k$ that is held fixed as the process proceeds to time grid points $t_{k+1}, t_{k+2}, \ldots$, may be continued until the terminal time is reached. This process is called the *self-starting Adams–Bashforth algorithm of order k*. For purposes of example calculations, it was implemented on a digital computer so that the effect of varying order $k$ of the algorithm and step size $h$ can be studied.

Detailed results for ten steps of Adams–Bashforth approximation of $e^t$ that are generated with a digital computer for $h = 0.1$ and $0.01$ are presented in Tables 7.4.6 and 7.4.7, respectively. Note that due to the self-starting procedure the results at $t_1$ are the same for all $k$. At $t_2$, the results are the same for $k = 2$ through 5. At $t_3$, the results are the same for $k = 3$ through 5. And so on. The substantial improvement in the accuracy of approximate solutions with increased $k$ and reduced step size, as predicted by the error estimate of Eq. 7.4.13, is also evident.

In Table 7.4.8, a summary of approximate solutions at $t = 10$, with $k = 1, 2, \ldots, 5$ and $h = 0.1$, is provided. Note that the error $E_k(10)$ in the approximate solution at $t = 10$ decreases as $k$ increases. These calculations are repeated with $h = 0.01$ and summarized in Table 7.4.9. As expected with a smaller step size, more accurate approximations are obtained. Recall, however, that 100 time steps are used with $h = 0.1$ and 1000 are required with $h = 0.01$, with an associated increase by a factor of 10 in computer cost. The solution is, however, 100 times more accurate, a nice return on investment, if accuracy is needed.

**TABLE 7.4.8 Adams–Bashforth Predictor Values of $e^{10}$, $h = 0.1$**

| $k$ | $x^P(10)$ | $E_k(10)$ | $\frac{\lvert E_k(10)\rvert}{e^{10}} \times 100(\%)$ |
|---|---|---|---|
| 1 | 13,780.612 | −8245.853 | 37 |
| 2 | 21,090.171 | −936.294 | 4 |
| 3 | 21,841.518 | −184.947 | 0.8 |
| 4 | 21,904.347 | −122.118 | 0.5 |
| 5 | 21,911.635 | −114.830 | 0.5 |

**TABLE 7.4.9 Adams–Bashforth Predictor Values of $e^{10}$, $h = 0.01$**

| $k$ | $x^P(10)$ | $E_k(10)$ | $\frac{\lvert E_k(10)\rvert}{e^{10}} \times 100(\%)$ |
|---|---|---|---|
| 1 | 20,959.155 | −1067.310 | 5 |
| 2 | 22,016.255 | −10.209 | 0.05 |
| 3 | 22,025.280 | −1.185 | 0.005 |
| 4 | 22,025.356 | −1.108 | 0.005 |
| 5 | 22,025.361 | −1.104 | 0.005 |

A brief examination of the results of Tables 7.4.8 and 7.4.9 reveals that with $h = 0.1$ the error in the approximation of the solution $e^{10}$ by the first-order algorithm ($k = 1$) is clearly unacceptable. With $h = 0.1$ and $k = 2$, as well as with $h = 0.01$ and $k = 1$, error is significant, but possibly acceptable for some applications. Acceptable solution accuracy, for many applications, is obtained with $h = 0.1$ and $k = 3$, 4, or 5. Excellent accuracy is obtained with $h = 0.01$ and $k = 3$, 4, or 5.

---

It is interesting to note that in Example 7.4.4, for $h = 0.01$, accuracy of $e^{10}$ is not significantly improved by increasing the order of the algorithm from $k = 4$ to $k = 5$. Similarly, for $h = 0.01$, accuracy is not significantly improved when the order is increased from $k = 3$ to $k = 4$ or 5. The reason for this apparent contradiction of the error estimate of Eq. 7.4.13 is that error in the starting steps, when $k = 1$ and 2, cannot be eliminated or compensated for by the much more accurate higher-order steps that occur throughout the remainder of the numerical integration process. This suggests that very small time steps should be used during the starting steps until enough time grid points with accurate solution values are available to switch to the intended larger time step and a higher-order algorithm. For example, a starting time step of $h_{\text{start}} = 0.01$ could be used for 40 time steps to reach $t = 0.4$ with order $k$ growing to 5. At that time data at $t = 0.0$, 0.1, 0.2, 0.3, and 0.4 could be used to implement the Adams–Bashforth algorithm with order $k = 5$ and $h = 0.1$ for the remainder of the numerical integration.

## 7.4.3 Adams–Moulton Corrector

The value $\mathbf{x}^p_{n+1}$ obtained by the Adams–Bashforth predictor may not be the best approximation to $\mathbf{x}(t_{n+1})$, because $\mathbf{f}(\mathbf{x}(t_{n+1}), t_{n+1})$ was not involved in the determination of $\mathbf{x}_{n+1}$. Indeed, if $\mathbf{f}(\mathbf{x}(t_{n+1}), t_{n+1})$ should unexpectedly change in going from $t_n$ to $t_{n+1}$, the difference between $\mathbf{x}_{n+1}$ and $\mathbf{x}(t_{n+1})$ could be much greater than $\boldsymbol{\tau}^p_{n+1}$. This is because the factor $\mathbf{x}^{(k+1)}(\xi)$ in Eq. 7.4.13, which previously was smooth, has suddenly changed in magnitude in the interval $t_n$ to $t_{n+1}$.

It seems plausible that a better approximation to $\mathbf{x}(t_{n+1})$ could be obtained if the Adams–Bashforth predicted value $\mathbf{x}^p_{n+1}$ of Eq. 7.4.11 were improved by incorporating it into an improved interpolating polynomial for $\mathbf{f}(\mathbf{x}(t), t)$ in Eq. 7.4.9. The *Adams–Moulton corrector formula of order* $k+1$ at $t_n$ uses a polynomial $\mathbf{P}^*_{k+1,n}(t)$ that interpolates the previous $k$ values of $\mathbf{f}_i$,

$$\mathbf{P}^*_{k+1,n}(t_{n+1-j}) = \mathbf{f}_{n+1-j}, \qquad j = 1, \ldots, k$$

and the approximation for $\mathbf{f}$ at $t_{n+1}$ that has been obtained from the Adams–Bashforth predictor; that is,

$$\mathbf{P}^*_{k+1,n}(t_{n+1}) = \mathbf{f}(\mathbf{x}^p_{n+1}, t_{n+1}) \equiv \mathbf{f}^p_{n+1}$$

Since $k+1$ points are being interpolated, $\mathbf{P}^*_{k+1,n}(t)$ is a polynomial of degree $k$.

Note that this is a polynomial of one higher degree than was used in the associated Adams–Bashforth predictor.

The approximate corrected solution $\mathbf{x}_{n+1}^c$ is obtained from Eq. 7.4.9 as

$$\mathbf{x}_{n+1}^c = \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{P}_{k+1,n}^*(t)\, dt \tag{7.4.14}$$

The backward difference form of Eq. 7.4.14, derived in exactly the same way as Eq. 7.4.11, yields the *Adams–Moulton corrector of order* $k+1$ [31] as

$$\mathbf{x}_{n+1}^c = \mathbf{x}_n + h \sum_{i=1}^{k+1} \gamma_{i-1}^* \nabla^{i-1} \mathbf{f}_{n+1}^p \tag{7.4.15}$$

where, with $s = (t - t_n)/h$,

$$\gamma_0^* = 1$$

$$\gamma_i^* = \frac{1}{i!} \int_0^1 (s-1)(s)\dots(s+i-2)\, ds, \qquad i \geqslant 1$$

and

$$\nabla^0 \mathbf{f}_{n+1}^p = \mathbf{f}_{n+1}^p$$

$$\nabla^1 \mathbf{f}_{n+1}^p = \nabla \mathbf{f}_{n+1}^p = \mathbf{f}_{n+1}^p - \mathbf{f}_n$$

$$\vdots$$

$$\nabla^i \mathbf{f}_{n+1}^p = \nabla^{i-1} \mathbf{f}_{n+1}^p - \nabla^{i-1} \mathbf{f}_n$$

The first five *Adams–Moulton coefficients* are given in Table 7.4.10.

**TABLE 7.4.10 Adams–Moulton Coefficients**

| $\gamma_0^*$ | $\gamma_1^*$ | $\gamma_2^*$ | $\gamma_3^*$ | $\gamma_4^*$ | $\gamma_5^*$ |
|---|---|---|---|---|---|
| 1 | $-\frac{1}{2}$ | $-\frac{1}{12}$ | $-\frac{1}{24}$ | $-\frac{19}{720}$ | $-\frac{3}{160}$ |

**Example 7.4.5:** Consider the initial-value problem of Example 7.4.4 for application of the Adams–Moulton corrector formula. The predicted value of $x_1$ is obtained with an Adams–Bashforth predictor of order 1, from Eq. 7.4.11, as

$$\begin{aligned} x_1^p &= x_0 + h(\gamma_0 \nabla^0 f_0) \\ &= 1 + (0.1)(1)(1) = 1.1 \end{aligned}$$

This prediction is corrected by the Adams–Moulton corrector of order 2; that is, from Eq. 7.4.15,

$$\begin{aligned} x_1^c &= x_0 + h(\gamma_0^* \nabla^0 f_1 + \gamma_1^* \nabla^1 f_1) \\ &= 1 + (0.1)\{(1)(1.1) + (-\tfrac{1}{2})(0.1)\} = 1.105 \end{aligned}$$

The value of $x_2$ is predicted with order 2 as

$$\begin{aligned} x_2^p &= x_1 + h(\gamma_0\nabla^0 f_1 + \gamma_1\nabla^1 f_1) \\ &= 1.105 + (0.1)\{(1)(1.105) = (0.5)(0.105)\} = 1.22075 \end{aligned}$$

and is corrected with order 3 as

$$\begin{aligned} x_2^c &= x_1 + h(\gamma_0^*\nabla^0 f_2 + \gamma_1^*\nabla^1 f_2 + \gamma_2^*\nabla^2 f_2) \\ &= 1.105 + (0.1)\{(1)(1.22075) + (-0.5)(0.11575) + (-\tfrac{1}{12})(0.01075)\} \\ &= 1.2211979 \end{aligned}$$

Similarly, the prediction of $x_3$ with order 3 is

$$\begin{aligned} x_3^p &= x_2 + h(\gamma_0\nabla^0 f_3 + \gamma_1\nabla^1 f_3 + \gamma_2\nabla^2 f_3) \\ &= 1.2211979 + (0.1)\{(1)(1.2211979) + (0.5)(0.1161979) + (\tfrac{5}{12})(0.0111979)\} \\ &= 1.3495942 \end{aligned}$$

and correction with order 4 yields

$$\begin{aligned} x_3^c &= x_2 + h(\gamma_0^*\Delta^0 f_3 + \gamma_1^*\Delta^1 f_3 + \gamma_2^*\Delta^2 f_3 + \gamma_3^*\Delta^3 f_3) \\ &= 1.2211979 + (0.1)\{(1)(1.3495942) + (-0.5)(0.1283963) \\ &\quad + (-\tfrac{1}{12})(0.0121984) + (-\tfrac{1}{24})(0.0010005)\} \\ &= 1.3496317 \end{aligned}$$

This sequence starts the *Adams–Bashforth, Adams–Moulton predictor–corrector algorithm,* which is self-starting.

In Table 7.4.11, a summary of the *predictor–corrector approximate solution* at $t = 10$, predicted by Adams–Bashforth formulas and corrected by one-higher-order Adams–Moulton formulas, is provided. The value $k$ given is the order of the Adams–Bashforth prediction that is used after starting is complete. The order of the Adams–Moulton corrector is one higher. Calculations are repeated with $h = 0.01$ and results are summarized in Table 7.4.12. Note that the solutions are substantially more accurate than those obtained in Example 7.4.4 using only Adams–Bashforth predictors.

**TABLE 7.4.11 Adams–Moulton Corrector Values of $e^{10}$, $h = 0.1$**

| $k$ | $x^c(10)$ | $E_k(10)$ | $\frac{\lvert E_k(10)\rvert}{e^{10}} \times 100(\%)$ |
|---|---|---|---|
| 1 | 21,688.414370387 | −338.051452637 | 2. |
| 2 | 21,980.499079835 | −45.966720581 | 0.2 |
| 3 | 21,994.743244024 | −31.722553253 | 0.2 |
| 4 | 21,990.708589885 | −35.757209778 | 0.2 |
| 5 | 21,986.702800628 | −39.763000488 | 0.2 |

**TABLE 7.4.12 Adams–Moulton Corrector Values of $e^{10}$, $h = 0.01$**

| $k$ | $x^c(10)$ | $E_k(10)$ | $\frac{\lvert E_k(10)\rvert}{e^{10}} \times 100(\%)$ |
|---|---|---|---|
| 1 | 22,022.822441367 | −3.643353462 | 0.02 |
| 2 | 22,026.251549431 | −0.214245379 | 0.001 |
| 3 | 22,026.188149769 | −0.277645051 | 0.001 |
| 4 | 22,026.130130982 | −0.335663855 | 0.002 |
| 5 | 22,026.088790048 | −0.377004802 | 0.002 |

Similar to the previous development, the local truncation error $\boldsymbol{\tau}^c_{n+1}$ of the corrected value is defined by

$$\mathbf{x}^c(t_{n+1}) = \mathbf{x}(t_n) + \sum_{i=1}^{k+1} \gamma^*_{i-1}\nabla^{i-1}\mathbf{f}^p_{n+1} + \boldsymbol{\tau}^c_{n+1}$$

It is shown in Reference 31 that, for constant step size $h$, this truncation error is given by

$$\boldsymbol{\tau}^c_{n+1} = \gamma^*_{k+1}h^{k+2}\mathbf{x}^{(k+2)}(\xi) \approx h\gamma^*_{k+1}\nabla^{k+1}\mathbf{f}^p_{n+1} \qquad \textbf{(7.4.16)}$$

with $\xi$ in $[t_{n-1}, t_{n+1}]$.

### 7.4.4 Computer Implementation of Predictor–Corrector Algorithms

The work involved in doing numerical integration on a computer is measured by the number of evaluations of $\mathbf{f}(\mathbf{x}, t)$ that are required. For computing intensive function evaluations, such as those encountered in mechanical system dynamics (see Sections 7.2 and 7.3), the remaining computation is relatively small, so this practice is an effective way of measuring work that is also machine independent. The predictor–corrector procedure costs two function evaluations at each step, so it is natural to ask, "Why not use the Adams–Bashforth method, which costs only one function evaluation at each step?" The predictor–corrector approach is more accurate and is so much better with respect to propagation of error that it can use steps that may be more than twice as large. Also, error estimates are more reliable, which leads to a more effective selection of step size.

The kind of predictor–corrector procedure presented here is called a *PECE method,* an acronym derived from a description of how the computation is done. First, Predict $\mathbf{x}^p_{n+1}$, Evaluate $\mathbf{f}^p_{n+1}$, Correct to obtain $\mathbf{x}^c_{n+1}$, and Evaluate $\mathbf{f}_{n+1}$ to complete the step. Corrector formulas of different orders could be considered, but in a PECE mode, there is no advantage in using a corrector formula that is more than one order higher than the predictor. It is shown in Reference 36 that a corrector of order $k+1$, together with a predictor of order $k$, is the best choice.

This choice is also convenient, since the predictor of order $k$ uses the values $\mathbf{x}_n$ and $\mathbf{f}_{n+1-j}$ for $j = 1, \ldots, k$, and the corrector of order $k+1$ uses the same values, along with $\mathbf{x}_{n+1}^p$. Thus, quantities to be retained are the same for the predictor and the corrector. Reference to PECE formulas of order $k$ will mean the Adams–Bashforth predictor of order $k$ and the Adams–Moulton corrector of order $k+1$.

Self-starting computer codes have been developed that require the use of only the differential equations and initial conditions. Since a PECE method requires two function evaluations per step, regardless of the order, and since higher-order formulas are expected to be more efficient, starting values for higher-order formulas are developed as in Examples 7.4.4 and 7.4.5. After completion of the first step with a first-order predictor and second-order corrector, $\mathbf{x}_1$, $\mathbf{f}_1$, and $\mathbf{f}_0$ have been stored, which is exactly the information that is needed for a second-order predictor and third-order corrector to take the second step. With $\mathbf{x}_2$, $\mathbf{f}_2$, $\mathbf{f}_1$, and $\mathbf{f}_0$, a third-order predictor and fourth-order corrector can be used, and so on. In this way, the order can be increased as necessary data are computed. The lower-order formulas are generally less accurate, which must be compensated for by taking smaller steps.

Excellent texts are available that expand on the theory outlined here for predictor–corrector solution of initial-value problems [31, 36, 37]. Error estimates are derived and their use in the selection of order $k$ and step size $h$ to maintain specified error tolerances is presented. A number of well-developed and tested computer codes are available that implement these techniques. One of the best of these codes is the DE numerical integration subroutine developed by Shampine and Gordon [36]. The reader who is interested in greater theoretical depth and information on the foundations of numerical integration computer codes is referred to References 36 and 38.

A few comments here may assist the reader in appreciating that the error estimates presented in Sections 7.4.2 and 7.4.3 can be used to adjust order $k$ and step size $h$ to satisfy an error tolerance. The error estimate of Eq. 7.4.16 is evaluated for the order $k$ and step size $h$ that have been used for the past few steps.

If components of $\boldsymbol{\tau}_{n+1}^c$ are greater than the specified error tolerance $\varepsilon > 0$, then the step size is reduced and/or the order $k$ is adjusted. Normally, step size is reduced by a factor of 2 and predictor calculations are repeated with the reduced step size. If the error estimate of Eq. 7.4.16 is still greater than $\varepsilon$, further action is required. The error estimate may now be evaluated for reduced and increased orders $k-1$ and $k+1$ to determine if a change in order will reduce error. If significant error reduction can be achieved by a modification of order, the change is made and integration is continued, perhaps with still further reduction in step size.

If all components of $\boldsymbol{\tau}_{n+1}^c$ after a predictor step are much smaller than $\varepsilon$, then the algorithm is too conservative and order and/or step size can be increased to achieve greater efficiency. As before, error estimates are evaluated with

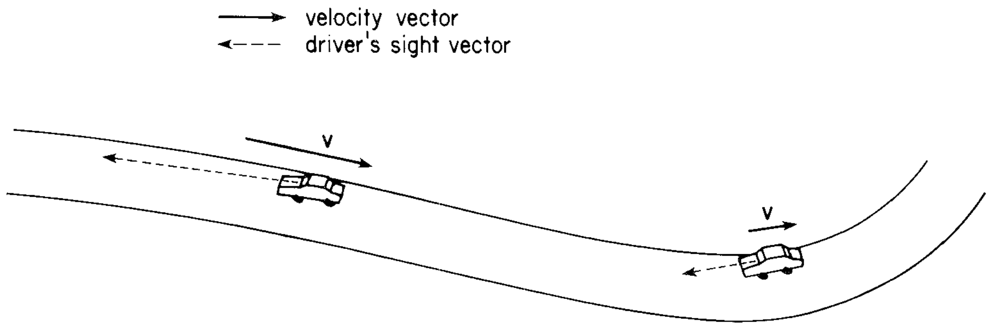

**Figure 7.4.2** Backward driving analogy.

reduced and increased order to determine if an order change is beneficial. Only if significant gains are possible are step size and/or order changed.

The process of changing step size and order may be intuitively related to the homely example of driving a vehicle backward, looking only at the road previously traversed. This analogy is meaningful, since numerical integration ahead in time in fact uses only past information. Along a straight stretch of road, as indicated at the left of Fig. 7.4.2, the driver may move with large velocity $v$, traversing long patches of road in a unit of time. This is analogous to using a large numerical step size $h$. At the same time, the driver is relatively relaxed and is viewing a long patch of the road over which he has come, as illustrated by the long dashed sight vector in Fig. 7.4.2. This is analogous to using a high integration order $k$, that is, making use of many previously passed data points.

If the road changes direction abruptly, as shown at the right of Fig. 7.4.2, the prudent driver will take two actions. First, the driver will slow down, which is analogous to reducing step size in the integration process. Second, the driver will concentrate his or her attention on road curvature information near the vehicle, as illustrated by the short dashed sight vector in Fig. 7.4.2, to sense the changing direction of the road. This is analogous to reducing the integration order $k$, using only the most recently traversed road data, which contains information on changing road direction.

## 7.5 NUMERICAL METHODS FOR EQUILIBRIUM ANALYSIS

A special mode of dynamic analysis that seeks to find a configuration in which velocity and acceleration are zero is called *equilibrium analysis*. The basic analytical condition that defines equilibrium is that $\dot{\mathbf{q}} = \ddot{\mathbf{q}} = \mathbf{0}$, consistent with the equations of motion of Eq. 6.4.2 and the equations of constraint. Three approaches to determining equilibrium configurations are discussed in this section, two of which are computationally practical. The first involves integrating the equations of motion of the system under the action of applied forces, perhaps adding artificial damping, until the system comes to rest at an equilibrium configuration. The second approach is based on writing the equations of motion with velocity and acceleration equal to zero and attempting to solve the resulting equations of equilibrium for an equilibrium configuration. The third and final method is based on the principle of minimum total potential energy for conservative systems, which states that a conservative system is in a state of stable equilibrium if and only if the total potential energy is at a strict relative minimum.

**Dynamic Settling** The most universal method for finding a stable equilibrium configuration for systems with nonconservative forces is to integrate the equations of motion until $\dot{\mathbf{q}} = \ddot{\mathbf{q}} = \mathbf{0}$ to within a numerical tolerance. To accelerate convergence to an equilibrium configuration, it is often desirable to add artificial damping to the system. Given a dynamic analysis capability, this method is easy to implement, but requires substantial computing time. In the case of dissipative nonconservative systems, many equilibrium configurations may exist. The one that actually occurs depends on the initial conditions chosen. In such situations, the dynamic settling approach to equilibrium analysis is the only valid method. Even when the system is nonconservative, but has a unique equilibrium configuration, the dynamic settling approach is valid and practical.

**Equations of Equilibrium** As noted previously, the analytical conditions for equilibrium are

$$\dot{\mathbf{q}} = \ddot{\mathbf{q}} = \mathbf{0} \tag{7.5.1}$$

Substituting these equations in the Lagrange multiplier form of the equations of motion of Eq. 7.2.1 yields the *system equilibrium equations*

$$\begin{aligned} \mathbf{\Phi}_{\mathbf{q}}^T(\mathbf{q})\boldsymbol{\lambda} &= \mathbf{Q}^A(\mathbf{q}) \\ \mathbf{\Phi}(\mathbf{q}) &= \mathbf{0} \end{aligned} \tag{7.5.2}$$

These equations are nonlinear in the generalized coordinates $\mathbf{q}$ and Lagrange multipliers $\boldsymbol{\lambda}$. Thus, some iterative form of numerical solution is required, such as Newton's method of Section 4.5.

Numerical solution of Eqs. 7.5.2 is not recommended for equilibrium analysis of general mechanical systems for several reasons. First, the Newton–Raphson equations that arise in attempting to solve this system are often ill-conditioned, leading to large perturbations and uncertainty as to convergence or arrival at an unwanted equilibrium configuration. This difficulty is aggravated by poor estimates of unknowns, which included Lagrange multipliers $\boldsymbol{\lambda}$. Intuition is often of little value in making reasonable estimates of $\boldsymbol{\lambda}$. Second, Eqs. 7.5.2 are satisfied at both stable and unstable equilibrium configurations. Therefore, depending on the estimate given to start Newton–Raphson iteration and conditioning of the Newton–Raphson equations, it is entirely possible that the algorithm will converge to an unstable equilibrium configuration that makes little physical sense, but is mathematically valid as a solution of Eqs. 7.5.2. This behavior has been observed to occur often for systems in which applied forces vary by several orders of magnitude, such as in systems with very heavy components and components of modest weight.

**Minimum Total Potential Energy** A well-known condition that defines a configuration $\mathbf{q}_e$ of stable equilibrium is that the *total potential energy* $V(\mathbf{q})$ of a *conservative system* should be at a strict relative minimum [34]; that is,

$$V(\mathbf{q}_e) < V(\mathbf{q}) \tag{7.5.3}$$

for all $\mathbf{q} \neq \mathbf{q}_e$ in a neighborhood of $\mathbf{q}_e$, such that

$$\mathbf{\Phi}(\mathbf{q}) = \mathbf{0} \tag{7.5.4}$$

This criterion for equilibrium is a *constrained minimization problem,* which can be solved using an algorithm similar to that employed for assembly analysis in Section 4.3. Prior to developing the minimization algorithm, however, an expression must be obtained for the total potential energy $V(\mathbf{q})$ of the system, and its gradient must be evaluated.

Consider first a constant force $\mathbf{F}$ that acts at point $P$ on a rigid body (e.g., a gravitational force). The virtual work of this force is

$$\delta W_F = \mathbf{F}^T \delta \mathbf{r}^P \tag{7.5.5}$$

Since force $\mathbf{F}$ is constant, this is the total differential of the work function

$$W_F = \mathbf{F}^T \mathbf{r}^P \tag{7.5.6}$$

Thus, the potential energy of the constant applied force $\mathbf{F}$ is

$$V_{\mathbf{F}} = -W_{\mathbf{F}} = -\mathbf{F}^T \mathbf{r}^P \tag{7.5.7}$$

where $\mathbf{r}^P$ depends on $\mathbf{q}$.

Consider next a translational spring–damper–actuator that acts between a pair of bodies, which applies a force that is a function of the length $\ell$ of the element,

$$f = k(\ell - \ell_0) + F(\ell) \tag{7.5.8}$$

where $\ell_0$ is the free length of the spring. Since a positive $f$ tends to draw the bodies together and a positive $\delta\ell$ tends to move them apart, the virtual work is

$$\begin{aligned} \delta W_f &= -f\,\delta\ell \\ &= -k(\ell - \ell_0)\,\delta\ell - F(\ell)\,\delta\ell \\ &= -k(\ell - \ell_0)\,\delta(\ell - \ell_0) - F(\ell)\,\delta\ell \end{aligned} \tag{7.5.9}$$

where $\delta\ell_0 = 0$, since $\ell_0$ is constant. Integrating Eq. 7.5.9, the work function associated with the translational spring–damper–actuator force is

$$W_f = -\frac{k}{2}(\ell - \ell_0)^2 - \int_{\ell_0}^{\ell} F(\ell)\,d\ell \tag{7.5.10}$$

Thus, the potential energy of the translational spring–damper–actuator force is simply

$$V_f = \frac{k}{2}(\ell - \ell_0)^2 + \int_{\ell_0}^{\ell} F(\ell)\, d\ell \tag{7.5.11}$$

Similarly, for a torsional spring–damper–actuator, the potential energy is

$$V_\tau = \frac{k_\theta}{2}(\theta - \theta_0)^2 + \int_{\theta_0}^{\theta} T(\theta)\, d\theta \tag{7.5.12}$$

The *total potential energy* of the system is the sum of the potential energies of all forces that act on the system; that is,

$$V = V_{\mathbf{F}} + V_f + V_\tau \tag{7.5.13}$$

where the sum is taken over all bodies in the system, so that $V = V(\mathbf{q})$.

For implementation of numerical minimization techniques, it is important to be able to calculate the derivatives of total potential energy with respect to generalized coordinates. It is well known in the theory of conservative force systems [34] that the gradient of total potential energy is the negative of the generalized applied force; that is,

$$V_{\mathbf{q}}^T = -\mathbf{Q}^A \tag{7.5.14}$$

Since the generalized applied force $\mathbf{Q}^A$ has been assembled in the formulation of the equations of motion, the gradient of total potential energy is readily available within the dynamics formulation presented in this text.

Many constrained optimization algorithms are available and can be applied to the minimization problem of Eqs. 7.5.3 and 7.5.4. However, the problem can be transformed to an unconstrained minimization problem using the generalized coordinate partitioning technique of Section 7.2.

For a mechanical system with a vector $\mathbf{q}$ of $nc$ generalized coordinates and $nh$ independent constraint equations $\mathbf{\Phi}(\mathbf{q}) = \mathbf{0}$, the vector $\mathbf{q}$ may be partitioned into $\mathbf{q} = [\mathbf{u}^T, \mathbf{v}^T]^T$, where $\mathbf{u}$ is an $nh$ vector of dependent generalized coordinates and $\mathbf{v}$ is an $nc - nh$ vector of independent generalized coordinates. Accordingly, the Jacobian $\mathbf{\Phi}_{\mathbf{q}}$ is partitioned into $\mathbf{\Phi}_{\mathbf{q}} = [\mathbf{\Phi}_{\mathbf{u}}, \mathbf{\Phi}_{\mathbf{v}}]$. The total differential of the constraint equations is

$$d\mathbf{\Phi} = \mathbf{\Phi}_{\mathbf{u}}\, d\mathbf{u} + \mathbf{\Phi}_{\mathbf{v}}\, d\mathbf{v} = \mathbf{0} \tag{7.5.15}$$

If the $nh$ constraint equations are independent, the submatrix $\mathbf{\Phi}_{\mathbf{u}}$ of $\mathbf{\Phi}_{\mathbf{q}}$ may be selected to be nonsingular. Therefore, Eq. 7.5.15 can be rewritten in the form

$$d\mathbf{u} = -\mathbf{\Phi}_{\mathbf{u}}^{-1}\mathbf{\Phi}_{\mathbf{v}}\, d\mathbf{v} \tag{7.5.16}$$

A matrix $\mathbf{H}$, called the *influence coefficient matrix*, is defined as

$$\mathbf{H} = -\mathbf{\Phi}_{\mathbf{u}}^{-1}\mathbf{\Phi}_{\mathbf{v}} \tag{7.5.17}$$

Then Eq. 7.5.16 becomes

$$d\mathbf{u} = \mathbf{H}\, d\mathbf{v} \tag{7.5.18}$$

The influence coefficient matrix $\mathbf{H}$ is calculated by solving

$$\mathbf{\Phi}_{\mathbf{u}}\mathbf{H}^{(i)} = -\mathbf{\Phi}_{\mathbf{v}}^{(i)}, \qquad i = 1, 2, \ldots, nc - nh \tag{7.5.19}$$

where $\mathbf{H}^{(i)}$ is the $i$th column of $\mathbf{H}$ and $\mathbf{\Phi}_{\mathbf{v}}^{(i)}$ is the $i$th column of $\mathbf{\Phi}_{\mathbf{v}}$.

The total differential of the potential energy function $V$ may be written as

$$dV = V_{\mathbf{u}}\, d\mathbf{u} + V_{\mathbf{v}}\, d\mathbf{v} \tag{7.5.20}$$

Substitution of Eq. 7.5.18 into Eq. 7.5.20 yields

$$\frac{dV}{d\mathbf{v}} = V_{\mathbf{u}}\mathbf{H} + V_{\mathbf{v}} \tag{7.5.21}$$

Recalling that the implicit function theorem guarantees that $\mathbf{u} = \mathbf{h}(\mathbf{v})$ (see Eq. 7.2.8), the minimization problem can be restated as choosing $\mathbf{v}$ to

$$\text{minimize } V = V(\mathbf{v}) \tag{7.5.22}$$

Equation 7.5.22 is an unconstrained optimization problem in the $nc - nh$ variables $\mathbf{v}$. Newton–Raphson iteration is used to determine dependent coordinates that satisfy the nonlinear kinematic constraint equations, while independent coordinates are fixed at the value determined by an unconstrained minimization algorithm at each step. The gradient $(dV/d\mathbf{v})$ is an $nc - nh$ vector that is needed for most commonly used optimization algorithms (e.g., the Fletcher–Powell algorithm of Section 4.3).

The gradient vector $dV^T/d\mathbf{v}$ can be easily determined at a feasible position $\mathbf{q}$, that is, where $\mathbf{\Phi}(\mathbf{q}) = \mathbf{0}$ is satisfied, as the negative of the generalized applied force $\mathbf{Q}^{AT}(\mathbf{q})$ from Eq. 7.5.14. If $\mathbf{Q}^A$ is partitioned, according to $\mathbf{q} = [\mathbf{u}^T, \mathbf{v}^T]^T$, into $\mathbf{Q}^A = [\mathbf{Q}^{A\mathbf{u}^T}, \mathbf{Q}^{A\mathbf{v}^T}]$, then

$$\begin{aligned} V_{\mathbf{u}}^T &= -\mathbf{Q}^{A\mathbf{u}} \\ V_{\mathbf{v}}^T &= -\mathbf{Q}^{A\mathbf{v}} \end{aligned} \tag{7.5.23}$$

Substituting Eq. 7.5.23 into Eq. 7.5.21 yields the desired result:

$$\frac{dV^T}{d\mathbf{v}} = -\mathbf{H}^T\mathbf{Q}^{A\mathbf{u}} - \mathbf{Q}^{A\mathbf{v}} \tag{7.5.24}$$

The Fletcher–Powell algorithm for unconstrained minimization presented in Section 4.3 may now be applied to minimize $V(\mathbf{v})$.

## PROBLEMS

### Section 7.2

**7.2.1.** Verify that Eq. 7.2.14 is valid.

**7.2.2.** Partition the Lagrange multiplier form of the equations of motion for the simple pendulum derived in Example 6.3.4 to obtain the result given in Example 7.2.1.

**7.2.3.** Verify that Eq. 7.2.18 is valid.

## Section 7.4

**7.4.1.** Show that $T_k^{(i)}(t_0) = f^{(i)}(t_0)$, where $T_k(t)$ is given by Eq. 7.4.2.

**7.4.2.** Show that $f(t_i) = P_k(t_i)$, where $P_k(t)$ is given by Eq. 7.4.4, for $k = 1$, 2, and 3, where $t_i = t_n - (n - i)h$, $i = n - k + 1, \dots, n$.

**7.4.3.** Make the change of variable $s = (t - t_n)/h$ in Eq. 7.4.12 and verify that the value of $\gamma_i$ is independent of $h$.