NASA Reference Publication 1327

Lawrence Livermore National Laboratory Report UCRL–ID–113855

1993

Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations

Krishnan Radhakrishnan
Sverdrup Technology, Inc.
Lewis Research Center Group

Alan C. Hindmarsh
Lawrence Livermore National Laboratory
Livermore, CA

National Aeronautics and Space Administration

Office of Management

Scientific and Technical Information Program

1993

Preface

This document provides a comprehensive description of LSODE, a solver for initial value problems in ordinary differential equation systems. It is intended to bring together numerous materials documenting various aspects of LSODE, including technical reports on the methods used, published papers on LSODE, usage documentation contained within the LSODE source, and unpublished notes on algorithmic details.

The three central chapters—on methods, code description, and code usage—are largely independent. Thus, for example, we intend that readers who are familiar with the solution methods and interested in how they are implemented in LSODE can read the Introduction and then chapter 3, Description of Code, without reading chapter 2, Description and Implementation of Methods. Similarly, those interested solely in how to use the code need read only the Introduction and then chapter 4, Description of Code Usage. In this case chapter 5, Example Problem, which illustrates code usage by means of a simple, stiff chemical kinetics problem, supplements chapter 4 and may be of further assistance.

Although this document is intended mainly for users of LSODE, it can be used as supplementary reading material for graduate and advanced undergraduate courses on numerical methods. Engineers and scientists who use numerical solution methods for ordinary differential equations may also benefit from this document.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

This report describes a FORTRAN subroutine package, LSODE, the Livermore Solver for Ordinary Differential Equations, written by Hindmarsh (refs. 1 and 2), and the methods included therein for the numerical solution of the initial value problem for a system of first-order ordinary differential equations (ODE's). Such a problem can be written as

$$\left.\begin{aligned} \dot{\underline{y}} &\equiv \frac{d\underline{y}}{d\xi} = \underline{f}(\underline{y}(\xi),\xi) \\ \underline{y}(\xi_0) &= \underline{y}_0 = \text{Given,} \end{aligned}\right\} \qquad (1.1)$$

where $\underline{y}$, $\underline{y}_0$, $\dot{\underline{y}}$, and $\underline{f}$ are column vectors with N (≥ 1) components and ξ is the independent variable, for example, time or distance. In component form equation (1.1) may be written as

$$\left.\begin{aligned} \frac{dy_i(\xi)}{d\xi} &= f_i(y_1(\xi),\ldots,y_N(\xi),\xi) \\ y_i(\xi_0) &= y_{i,0} = \text{Given} \end{aligned}\right\} \quad i = 1,\ldots,N. \qquad (1.2)$$

The initial value problem is to find the solution function $\underline{y}$ at one or more values of ξ in a prescribed integration interval $[\xi_0,\xi_{\text{end}}]$, where the initial value of $\underline{y}$, $\underline{y}_0$, at $\xi = \xi_0$ is given. The endpoint, ξ_{end}, may not be known in advance as, for example, when asymptotic values of $\underline{y}$ as $\xi \to \infty$ are required.

Initial value, first-order ODE's arise in many fields, such as chemical kinetics, biology, electric network analysis, and control theory. It is assumed that the problem is well posed and possesses a solution that is unique in the interval of interest. Solution existence and uniqueness are guaranteed if, in the region of interest, $\underline{f}$ is defined and continuous and for any two vectors $\underline{y}$ and $\underline{y}^*$ in that region there exists a positive constant $\mathcal{L}$ such that (refs. 3 and 4)

$$\left\| \underline{f}(\underline{y},\xi) - \underline{f}(\underline{y}^*,\xi) \right\| \leq \mathcal{L} \left\| \underline{y} - \underline{y}^* \right\|, \tag{1.3}$$

which is known as a Lipschitz condition. Here $\|\bullet\|$ denotes a vector norm (e.g., ref. 5), and the constant $\mathcal{L}$ is known as a Lipschitz constant of $\underline{f}$ with respect to $\underline{y}$.

The right-hand side $\underline{f}$ of the ODE system must be a function of $\underline{y}$ and ξ only. It cannot therefore involve $\underline{y}$ at previous ξ values, as in delay or retarded ODE's or integrodifferential equations. It cannot also involve random variables, as in stochastic differential equations. A second- or higher-order ODE system must be reduced to a first-order ODE system.

The solution methods included in LSODE replace the ODE's with difference equations and then solve them step by step. Starting with the initial conditions at ξ_0, approximations $\underline{Y}_n (= Y_{i,n}, i = 1,...,N)$ to the exact solution $\underline{y}(\xi_n)$ $[= y_i(\xi_n), i = 1,...,N]$ of the ODE's are generated at the discrete mesh points ξ_n $(n = 1,2,...)$, which are themselves determined by the package. The spacing between any two mesh points is called the step size or step length and is denoted by h_n, where

$$h_n = \xi_n - \xi_{n-1}. \tag{1.4}$$

An important feature of LSODE is its capability of solving "stiff" ODE problems. For reasons discussed by Shampine (ref. 6) stiffness does not have a simple definition involving only the mathematical problem, equation (1.1). However, Shampine and Gear (ref. 7) discuss some fundamental issues related to stiffness and how it arises. An approximate description of a stiff ODE system is that it contains both very rapidly and very slowly decaying terms. Also, a characteristic of such a system is that the $N \times N$ Jacobian matrix $\mathbf{J}$ $(= \partial\underline{f}/\partial\underline{y})$, with element J_{ij} defined as

$$J_{ij} = \partial f_i/\partial y_j, \qquad i,j = 1,\ldots,N, \tag{1.5}$$

has eigenvalues $\{\lambda_i\}$ with real parts that are predominantly negative and also vary widely in magnitude. Now the solution varies locally as a linear combination of the exponentials $\{e^{\xi \mathrm{Re}(\lambda_i)}\}$, which all decay if all $\mathrm{Re}(\lambda_i) < 0$, where $\mathrm{Re}(\lambda_i)$ is the real part of λ_i. Hence for sufficiently large ξ ($> 1/\max|\mathrm{Re}(\lambda_i)|$, where the bars $|\bullet|$ denote absolute value), the terms with the largest $\mathrm{Re}(\lambda_i)$ will have decayed to insignificantly small levels while others are still active, and the problem would be classified as stiff. If, on the other hand, the integration interval is limited to $1/\max|\mathrm{Re}(\lambda_i)|$, the problem would not be considered stiff.

In this discussion we have assumed that all eigenvalues have negative real parts. Some of the $\mathrm{Re}(\lambda_i)$ may be nonnegative, so that some solution components are nondecaying. However, the problem is still considered stiff if no eigenvalue has a real part that is both positive and large in magnitude and at least one eigenvalue has a real part that is both negative and large in magnitude (ref. 7). Because the $\{\lambda_i\}$ are, in general, not constant, the property of stiffness is local in that a problem may be stiff in some intervals and not in others. It is also relative in the sense that one problem may be more stiff than another. A quantitative measure of stiffness is usually given by the stiffness ratio $\max[-\mathrm{Re}(\lambda_i)]/\min[-\mathrm{Re}(\lambda_i)]$. This measure is also local for the reason given previously. Another standard measure for stiffness is the quantity $\max[-\mathrm{Re}(\lambda_i)]|\xi_{\mathrm{end}} - \xi_0|$. This measure is more relevant than the previous one when $|\xi_{\mathrm{end}} - \xi_0|$ is a better indicator of the average "resolution scale" for the problem than $1/\min[-\mathrm{Re}(\lambda_i)]$. (In some cases $\min[-\mathrm{Re}(\lambda_i)] = 0$.)

The difficulty with stiff problems is the prohibitive amounts of computer time required for their solution by classical ODE solution methods, such as the popular explicit Runge-Kutta and Adams methods. The reason is the excessively small step sizes that these methods must use to satisfy stability requirements. Because of the approximate nature of the solutions generated by numerical integration methods, errors are inevitably introduced at every step. For a numerical method to be stable, errors introduced at any one step should not grow unbounded as the calculation proceeds. To maintain numerical stability, classical ODE solution methods must use small step sizes of order $1/\max[-\mathrm{Re}(\lambda_i)]$ even after the rapidly decaying components have decreased to negligible levels. Examples of the step size pattern used by an explicit Runge-Kutta method in solving stiff ODE problems arising in combustion chemistry are given in references 8 and 9. Now, the size of the integration interval for the evolution of the slowly varying components is of order $1/\min[-\mathrm{Re}(\lambda_i)]$. Consequently, the number of steps required by classical methods to solve the problem is of order $\max[-\mathrm{Re}(\lambda_i)]/\min[-\mathrm{Re}(\lambda_i)]$, which is very large for stiff ODE's.

For stiff problems the LSODE package uses the backward differentiation formula (BDF) method (e.g., ref. 10), which is among the most popular currently used for such problems (ref. 11). The BDF method possesses the property of stiff stability (ref. 10) and therefore does not suffer from the stability step size constraint once the rapid components have decayed to negligible levels. Throughout the integration the step size is limited only by accuracy requirements imposed on the numerical solution. Accuracy of a numerical method refers to the magnitude of the error introduced in a single step or, more precisely, the local truncation or discretization error. The local truncation error $\underline{d}_n$ at ξ_n is the difference between the computed approximation and the exact solution, with both starting the integration at the previous mesh point ξ_{n-1} and using the exact solution $\underline{y}(\xi_{n-1})$ as the initial value. The local truncation error on any step is therefore the error incurred on that step under the assumption of no past errors (e.g., ref. 12).

The accuracy of a numerical method is usually measured by its order. A method is said to be of order q if the local truncation error varies as h_n^{q+1}. More precisely, a numerical method is of order q if there are quantities $\underline{C}$ and h_o (> 0) such that (refs. 3 and 13)

$$|\underline{d}_n| \le \underline{C}\, h_n^{q+1} \qquad \text{for all} \qquad 0 < h_n \le h_o, \tag{1.6}$$

where $|\underline{d}_n|$ is an N-dimensional column vector containing the absolute values of the $d_{i,n}$ ($i = 1,...,N$). The coefficient vector $\underline{C}$ may depend on the function defining the ODE and the total integration interval, but it should be independent of the step size h_n (ref. 13). Accuracy of a numerical method refers to the smallness of the error introduced in a single step; stability refers to whether or not this error grows in subsequent steps (ref. 7).

To satisfy accuracy requirements, the BDF method may have to use small step sizes of order 1/max(Re $|\lambda_i|$) in regions where the most rapid exponentials are active. However, outside these regions, which are usually small relative to the total integration interval, larger step sizes may be used.

The LSODE package also includes the implicit Adams method (e.g., refs. 4 and 10), which is well suited for nonstiff problems. Both integration methods belong to the family of linear multistep methods. As implemented in LSODE these methods allow both the step size and the method order to vary (from 1 to 12 for the Adams method and from 1 to 5 for the BDF method) throughout the problem. The capability of dynamically varying the step size and the method order is very important to the efficient use of linear multistep methods (ref. 14).

The LSODE package consists of 21 subprograms and a BLOCK DATA module. The package has been designed to be used as a single unit, and in normal circumstances the user needs to communicate with only a single subprogram, also called LSODE for convenience. LSODE is based on, and in many ways resembles, the package GEAR (ref. 15), which, in turn, is based on the code DIFSUB, written by Gear (refs. 10 and 16). All three codes use integration methods that are based on a constant step size but are implemented in a manner that allows for the step size to be dynamically varied throughout the problem. There are, however, many differences between GEAR and LSODE, with the following important improvements in LSODE over GEAR: (1) its user interface is much more flexible; (2) it is more extensively modularized; and (3) it uses dynamic storage allocation, different linear algebra modules, and a wider range of error types (ref. 17). Most significantly, LSODE has been designed to virtually eliminate the need for user adjustments or modifications to the package before it can be used effectively. For example, the use of dynamic storage allocation means that the required total storage is specified once in the user-supplied subprogram that communicates with LSODE; there is no need to adjust any dimension declarations in the package. This feature, besides making the code easy to use, minimizes the total storage requirements; only the storage required for the user's problem needs to be allocated and not that called for by a code using default values for parameters, such as the total number of ODE's, for example. The many different capabilities of the code can be exploited quite simply by setting values for appropriate parameters in the user's subprogram. Not requiring any adjustments to the code eliminates the user's need to become familiar with the inner workings of the code, which can therefore be used as a "black box," and, more importantly, eliminates the possibility of errors being introduced into the modified version.

The remainder of this report is organized as follows: In chapter 2 we describe the numerical integration methods used in LSODE and how they are implemented in practice. The material presented in this chapter is based on, and closely follows, the developments by Gear (refs. 10 and 18 to 20) and Hindmarsh (refs. 1, 2, 15, 21, and 22). Chapter 3 describes the features and layout of the LSODE package. In chapter 4 we provide a detailed guide to its usage, including possible user modifications. The use of the code is illustrated by means of a simple test problem in chapter 5. We conclude this report with a brief discussion on code availability in chapter 6.

Chapter 2 Description and Implementation of Methods

2.1 Linear Multistep Methods

The numerical methods included in the packaged code LSODE generate approximate solutions $\underline{Y}_n$ to the ordinary differential equations (ODE's) at discrete points ξ_n ($n = 1,2,...$). Assuming that the approximate solutions $\underline{Y}_{n-j}$ have been computed at the mesh points ξ_{n-j} ($j = 1,2,...$), these methods advance the solution to the current value ξ_n of the independent variable by using linear multistep formulas of the type

$$\underline{Y}_n = \sum_{j=1}^{K_1} \alpha_j \underline{Y}_{n-j} + h_n \sum_{j=0}^{K_2} \beta_j \underline{f}_{n-j}, \tag{2.1}$$

where the current approximate solution vector $\underline{Y}_n$ consists of N components,

$$\underline{Y}_n = \left(Y_{1,n},\ldots,Y_{N,n}\right)^T, \tag{2.2}$$

and the superscript T indicates transpose. In equation (2.1), $\underline{f}_{n-j}$ [$= \underline{f}(\underline{Y}_{n-j})$] is the approximation to the exact derivative vector at ξ_{n-j}, $\dot{\underline{y}}(\xi_{n-j})$ [$= \underline{f}(\underline{y}(\xi_{n-j}))$], where for notational convenience the ξ argument of $\underline{f}$ has been dropped; the coefficients $\{\alpha_j\}$ and $\{\beta_j\}$ and the integers K_1 and K_2 are associated with a particular method; and h_n ($= \xi_n - \xi_{n-1}$) is the step size to be attempted on the current step $[\xi_{n-1},\xi_n]$. The method is called linear because the $\{\underline{Y}_j\}$ and $\{\underline{f}_j\}$ occur linearly. It is called multistep because it uses information from several previous mesh points. The number $\max(K_1, K_2)$ gives the number of previous values involved.

The values $K_1 = 1$ and $K_2 = q - 1$ produce the popular implicit Adams, or Adams-Moulton (AM), method of order q:

$$\underline{Y}_n = \underline{Y}_{n-1} + h_n \sum_{j=0}^{q-1} \beta_j \underline{f}_{n-j}. \tag{2.3}$$

The method is called implicit because it uses the as yet unknown $\underline{f}_n$ to compute $\underline{Y}_n$. The method order q means that if equation (2.3) is solved with all past values being exact, the resulting $\underline{Y}_n$ will differ from the exact solution $\underline{y}(\xi_n)$ to the ODE system by a local truncation error that is of order $O(H^{q+1})$ for small values of $H = \max|h_k|$.

The choice $K_1 = q$, $K_2 = 0$ results in the backward differentiation formula (BDF) method of order q:

$$\underline{Y}_n = \sum_{j=1}^{q} \alpha_j \underline{Y}_{n-j} + h_n \beta_0 \underline{f}_n. \tag{2.4}$$

The term "backward differentiation formula" is used to describe the method because equation (2.4), upon division by $h_n\beta_0$ and rearrangement of terms, can be regarded as an approximation for $\underline{\dot{y}}(\xi_n)$ in terms of $\underline{Y}_n, \underline{Y}_{n-1}, \ldots, \underline{Y}_{n-q}$ (refs. 15 and 17).

The two methods can be written in the general form

$$\underline{Y}_n = \underline{\psi}_n + h_n\beta_0 \underline{f}_n = \underline{\psi}_n + h_n\beta_0 \underline{f}(\underline{Y}_n), \tag{2.5}$$

where $\underline{\psi}_n$ contains previously computed information and is given by

$$\underline{\psi}_n = \underline{Y}_{n-1} + h_n \sum_{j=1}^{q-1} \beta_j \underline{f}_{n-j} \tag{2.6a}$$

for the AM method of order q, and

$$\underline{\psi}_n = \sum_{j=1}^{q} \alpha_j \underline{Y}_{n-j} \tag{2.6b}$$

for the BDF method of order q.

The coefficients $\{\alpha_j\}$ and $\{\beta_j\}$ are determined such that equations (2.3) and (2.4) will be exact if the solution to equation (1.1) is a polynomial of degree q or less. Stability characteristics limit q in equation (2.4) to 6 (ref. 10). In LSODE, however, BDF's of order up to only 5 are used because of additional stability considerations (refs. 7 and 23). The coefficients $\{\alpha_j\}$ and $\{\beta_j\}$ for the two methods are given by Gear (ref. 10) for $q \le 6$. In equation (2.5), although the subscript n has been attached to the step size h, indicating that h_n is the step size to be attempted on the current step, the methods used in LSODE are based on a constant h. When the step size is changed, the data at the new spacing required to continue the integration are obtained by interpolating from the data at the original spacing. Solution methods and codes that are based on variable step size have also been developed (refs. 17, 23, and 24) but are not considered in the present work.

2.2 Corrector Iteration Methods

If $\beta_0 = 0$, the methods are called explicit because they involve only the known values $\{\underline{Y}_{n-j}\}$ and $\{\underline{f}_{n-j}\}$, and equation (2.1) is easy to solve. If, however, $\beta_0 \neq 0$, the methods are called implicit and, in general, solution of equation (2.1) is expensive. For both methods, equations (2.3) and (2.4), β_0 is positive for each q and because $\underline{f}$ is, in general, nonlinear, some type of iterative procedure is needed to solve equation (2.5). Nevertheless, implicit methods are preferred because they are more stable, and hence can use much larger step sizes, than explicit methods and are also more accurate for the same order and step size (refs. 4, 10, and 12). Explicit methods are used as predictors, which generate an initial guess for $\underline{Y}_n$. The implicit method corrects the initial guess iteratively and provides a reasonable approximation to the solution of equation (2.5).

The predictor-corrector process for advancing the numerical solution to ξ_n therefore consists of first generating a predicted value, denoted by $\underline{Y}_n^{[0]}$, and then correcting this initial estimate by iterating equation (2.5) to convergence. That is, starting with the initial guess $\underline{Y}_n^{[0]}$, approximations $\underline{Y}_n^{[m]}$ $(m = 1,2,...,M)$ are generated (by using one of the techniques discussed below) until the magnitude of the difference in two successive approximations approaches zero within a specified accuracy. The quantity $\underline{Y}_n^{[m]}$ is the approximation obtained on the mth iteration, the integer M is the number of iterations required for convergence, and we accept $\underline{Y}_n^{[M]}$ as an approximation to the exact solution $\underline{y}$ at ξ_n and therefore denote it by $\underline{Y}_n$ although, in general, it does not satisfy equation (2.5) exactly.

At each iteration m the quantity $h_n\dot{\underline{Y}}_n^{[m]}$, which is defined here, is computed from $\underline{Y}_n^{[m]}$ by the relation

$$\underline{Y}_n^{[m]} = \underline{\psi}_n + \beta_0 h_n \dot{\underline{Y}}_n^{[m]}. \tag{2.7}$$

Now, as discussed by Hindmarsh (ref. 21) and shown later in this section, if $\underline{Y}_n^{[m]}$ converges as $m \to \infty$, the limit, that is, $\lim_{m\to\infty} \underline{Y}_n^{[m]}$, must be a solution of equation (2.5) and $\dot{\underline{Y}}_n^{[m]}$ converges to $\underline{f}_n$ [= $\underline{f}(\underline{Y}_n)$], the approximation to $\dot{\underline{y}}(\xi_n)$.

Hence $h_n\dot{\underline{Y}}_n^{[m]}$ is the mth estimate for $h_n\underline{f}_n$ and $\lim_{m\to\infty} h_n\dot{\underline{Y}}_n^{[m]} = h_n\underline{f}_n$. The predicted value of $h_n\underline{f}_n$, denoted by $h_n\dot{\underline{Y}}_n^{[0]}$, is also obtained from equation (2.7) (by setting $m = 0$). In practice, we terminate the calculation sequence at a finite number M of iterations and accept as an approximation to $h_n\underline{f}_n$ the quantity $h_n\dot{\underline{Y}}_n \equiv h_n\dot{\underline{Y}}_n^{[M]}$, which is obtained from $\underline{Y}_n^{[M]}$ by using equation (2.7). Note that $\dot{\underline{Y}}_n$ is only an approximation to $\underline{f}_n$ because $\underline{Y}_n^{[M]}$ does not, in general, satisfy equation (2.5) exactly (see eqs. (2.5) and (2.7)). Moreover, because $\dot{\underline{Y}}_n^{[M]}$ is defined to satisfy the solution method, in the sense of equation (2.7), it is not necessarily equal to $\underline{f}(\underline{Y}_n^{[M]})$. Therefore $\underline{Y}_n^{[M]}$ and $\dot{\underline{Y}}_n^{[M]}$ do not necessarily satisfy the ODE, equation (1.1). Thus, in practice, to advance the solution, the methods use the $\{\dot{\underline{Y}}_j\}$ (e.g., see eqs. (2.8a) and (2.8b)), rather than the $\{\underline{f}_j\}$ as written in equation (2.1).

After convergence of the estimates $\underline{Y}_n^{[m]}$, we could define $\dot{\underline{Y}}_n^{[M]}$ to be equal to $\underline{f}(\underline{Y}_n^{[M]})$, so that $\underline{Y}_n^{[M]}$ and $\dot{\underline{Y}}_n^{[M]}$ satisfy the ODE exactly. However, besides being more expensive because it will require one derivative evaluation, performing this operation is actually less stable for stiff equations than using equation (2.7) (ref. 25).

The predicted value at ξ_n, $\underline{Y}_n^{[0]}$, is generated by a qth-order explicit formula similar to equations (2.3) and (2.4) (refs. 18 and 20):

$$\underline{Y}_n^{[0]} = \underline{Y}_{n-1} + h_n \sum_{j=1}^{q} \beta_j^* \dot{\underline{Y}}_{n-j} \tag{2.8a}$$

for the AM method of order q and

$$\underline{Y}_n^{[0]} = \sum_{j=1}^{q} \alpha_j^* \underline{Y}_{n-j} + h_n \beta_1^* \dot{\underline{Y}}_{n-1} \tag{2.8b}$$

for the BDF method of order q. In these two equations $\dot{\underline{Y}}_{n-j}$ is the approximation to $\underline{f}_{n-j}$ computed on the step $[\xi_{n-j-1},\xi_{n-j}]$. The coefficients $\{\alpha_j^*\}$ and $\{\beta_j^*\}$ are selected such that equation (2.8a) or (2.8b) will be exact if the solution to equation (1.1) is a polynomial of degree q or less.

The predictor step for the two methods can be generalized trivially as

$$\underline{Y}_n^{[0]} = \underline{\psi}_n^*, \tag{2.9}$$

where $\underline{\psi}_n^*$ is given by the right-hand sides of equations (2.8a) and (2.8b), respectively, for the AM and BDF methods.

To correct the initial estimate given by equation (2.9), that is, to solve equation (2.5), LSODE includes a variety of iteration techniques—functional, Newton-Raphson, and a variant of Jacobi-Newton.

2.2.1 Functional Iteration

To derive the functional iteration technique, also called simple iteration (refs. 11 and 26) and successive substitution (ref. 27), we rewrite equation (2.5) as follows:

$$\underline{Y}_n = \underline{\Phi}(\underline{Y}_n), \tag{2.10}$$

where

$$\underline{\Phi}(\underline{Y}_n) = \underline{\psi}_n + h_n\beta_0\underline{f}(\underline{Y}_n). \tag{2.11}$$

The $(m + 1)$th estimate, $\underline{Y}_n^{[m+1]}$ $(m = 0,1,\ldots,M-1)$, is then obtained from equation (2.10) by (e.g., ref. 27)

$$\underline{Y}_n^{[m+1]} = \underline{\Phi}\left(\underline{Y}_n^{[m]}\right) = \underline{\psi}_n + h_n\beta_0\underline{f}\left(\underline{Y}_n^{[m]}\right). \tag{2.12}$$

Now equation (2.7) gives the following expression for $h_n\underline{\dot{Y}}_n^{[m+1]}$:

$$\underline{Y}_n^{[m+1]} = \underline{\psi}_n + \beta_0 h_n\underline{\dot{Y}}_n^{[m+1]}. \tag{2.13}$$

Comparing equations (2.12) and (2.13) gives

$$h_n\underline{\dot{Y}}_n^{[m+1]} = h_n\underline{f}\left(\underline{Y}_n^{[m]}\right) \tag{2.14}$$

for functional iteration.

We now define the vector function $\underline{g}(\underline{y})$ by

$$\underline{g}(\underline{y}) = h_n\underline{f}(\underline{y}) + \frac{\underline{\psi}_n - \underline{y}}{\beta_0}, \tag{2.15}$$

which, upon using equation (2.7), gives

$$\underline{g}\left(\underline{Y}_n^{[m]}\right) = h_n \underline{f}\left(\underline{Y}_n^{[m]}\right) - h_n \underline{\dot{Y}}_n^{[m]}. \tag{2.16}$$

By using equation (2.15) we can rewrite the functional iteration equation (2.12) as follows:

$$\underline{Y}_n^{[m+1]} = \underline{Y}_n^{[m]} + \beta_0 \underline{g}\left(\underline{Y}_n^{[m]}\right). \tag{2.17}$$

Finally the combination of equations (2.14) and (2.16) produces the following functional iteration procedure for $h_n \underline{\dot{Y}}_n$:

$$h_n \underline{\dot{Y}}_n^{[m+1]} = h_n \underline{\dot{Y}}_n^{[m]} + \underline{g}\left(\underline{Y}_n^{[m]}\right). \tag{2.18}$$

Equation (2.17) is simple to use, but it converges only linearly (ref. 27). In addition, for successful convergence the step size may be restricted to very small values for stiff problems (refs. 4, 10, 12, 26, and 28), as shown here. By using equation (2.14) we can rewrite equation (2.16) as

$$\underline{g}\left(\underline{Y}_n^{[m]}\right) = h_n \underline{f}\left(\underline{Y}_n^{[m]}\right) - h_n \underline{f}\left(\underline{Y}_n^{[m-1]}\right), \tag{2.19}$$

for $m \geq 1$. Hence, equation (2.17) can be rewritten as

$$\underline{Y}_n^{[m+1]} = \underline{Y}_n^{[m]} + h_n \beta_0 \left[\underline{f}\left(\underline{Y}_n^{[m]}\right) - \underline{f}\left(\underline{Y}_n^{[m-1]}\right)\right]. \tag{2.20}$$

By using the Lipschitz condition, equation (1.3), we get the following relation from equation (2.20):

$$\left\|\underline{Y}_n^{[m+1]} - \underline{Y}_n^{[m]}\right\| \leq \left|h_n\right| \beta_0 \mathcal{L} \left\|\underline{Y}_n^{[m]} - \underline{Y}_n^{[m-1]}\right\|, \tag{2.21}$$

which shows that the iteration converges, that is, the successive differences

$$\left\|\underline{Y}_n^{[m+1]} - \underline{Y}_n^{[m]}\right\|$$

decrease, only if

$$|h_n|\beta_0\mathcal{L} < 1. \tag{2.22}$$

Now stiff problems are characterized by, and often referred to as systems with, large Lipschitz constants (e.g., refs. 4, 12, and 26), and so equation (2.22) restricts the step size to very small values. Indeed, the restriction imposed by this inequality on h_n is exactly of the same form as that imposed by stability requirements on classical methods, such as the explicit Runge-Kutta method (refs. 4 and 26). For this reason, when functional iteration is used, the integration method is usually said to be explicit even though it is implicit (ref. 17).

2.2.2 Newton-Raphson Iteration

Newton-Raphson (NR) iteration, on the other hand, converges quadratically and can use much larger step sizes than functional iteration (refs. 27, 29, and 30). Rapid improvement in the accuracy of the estimates is especially important because the corrector is iterated to convergence. The reason for iterating to convergence is to preserve the stability characteristics of the corrector. If the correction process is terminated after a fixed number of iterations, the stability characteristics of the corrector are lost (refs. 4 and 12), with disastrous consequences for stiff problems.

To derive the NR iteration procedure, we rewrite equation (2.5) as

$$\underline{R}(\underline{Y}_n) = \underline{Y}_n - \underline{\psi}_n - h_n\beta_0\underline{f}(\underline{Y}_n) = 0, \tag{2.23}$$

so that solving equation (2.5) is equivalent to finding the zero of $\underline{R}$. The quantity $\underline{R}(\underline{Y}_n^{[m]})$ is the residual vector on the mth iteration; that is, it is the amount by which $\underline{Y}_n^{[m]}$ fails to satisfy equation (2.5). To obtain the (m + 1)th estimate, we expand equation (2.23) in a Taylor series about the mth estimate, neglect the second and higher derivatives, and set $\underline{R}(\underline{Y}_n^{[m+1]}) = 0$ because we seek a $\underline{Y}_n^{[m+1]}$ that produces this result (e.g., ref. 27). Performing these operations and then rearranging terms give the following relation for the NR iteration technique:

$$\mathbf{P}\left(\underline{Y}_n^{[m+1]} - \underline{Y}_n^{[m]}\right) = -\underline{R}\left(\underline{Y}_n^{[m]}\right) = \underline{\psi}_n + h_n\beta_0\underline{f}\left(\underline{Y}_n^{[m]}\right) - \underline{Y}_n^{[m]}, \tag{2.24}$$

where the $N\times N$ matrix $\mathbf{P}$ is given by

$$\mathbf{P} = \partial\underline{R}/\partial\underline{Y} = \mathbf{I} - h_n\beta_0\mathbf{J}. \tag{2.25}$$

In equation (2.25), **I** is the $N \times N$ identity matrix and **J** is the Jacobian matrix, equation (1.5). Comparing equations (2.15) and (2.23) shows that

$$\underline{R}(\underline{Y}) = -\beta_0 \underline{g}(\underline{Y}), \tag{2.26}$$

so that equation (2.24) can be rewritten as follows:

$$\underline{Y}_n^{[m+1]} = \underline{Y}_n^{[m]} + \beta_0 \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right). \tag{2.27}$$

The NR iteration procedure for $h_n \dot{\underline{Y}}_n$ is derived by subtracting equation (2.7) from equation (2.13) and then using equation (2.27). The result is

$$h_n \dot{\underline{Y}}_n^{[m+1]} = h_n \dot{\underline{Y}}_n^{[m]} + \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right). \tag{2.28}$$

This iteration will converge provided that the predicted value is sufficiently accurate (refs. 4 and 12). The prediction method, equation (2.9), provides a sufficiently accurate initial estimate that the average number of iterations per step is less than 1.5 (ref. 7). In fact, the predictor is generally as accurate as the corrector, which is nonetheless needed for numerical stability. However, much computational work is required to form the Jacobian matrix and to perform the linear algebra necessary to solve equation (2.27). Now, because the Jacobian does not appear explicitly in the ODE's, equation (1.1), or in the solution method, equation (2.5), **J** need not be very accurate. Therefore, for problems in which the analytical Jacobian matrix is difficult or impossible to evaluate, a fairly crude approximation such as the finite-difference quotient

$$J_{ij} = \frac{f_i\left(Y_j + \Delta Y_j\right) - f_i\left(Y_j\right)}{\Delta Y_j}, \quad i, j = 1, \ldots, N, \tag{2.29}$$

is adequate. In equation (2.29), ΔY_j is a suitable increment for the jth component of $\underline{Y}$.

Inaccuracies in the iteration matrix may affect the rate of convergence of the solution but not the solution if it converges (refs. 4 and 21). Hence this matrix need only be accurate enough for the iteration to converge. This beneficial fact can be used to reduce the computational work associated with linear algebra, as described in chapter 3.

2.2.3 Jacobi-Newton Iteration

Jacobi-Newton (JN) iteration (ref. 31), also called Jacobi iteration (ref. 32), is obtained from Newton-Raphson iteration by neglecting all off-diagonal elements of the Jacobian matrix. Hence for JN iteration

$$J_{ij} = \begin{cases} 0, & i \neq j \\ \partial f_i / \partial y_j, & i = j. \end{cases} \tag{2.30}$$

This technique is as simple to use as functional iteration because it does not require any matrix algebra. Also, it converges faster than functional iteration but, in general, not as fast as NR iteration.

A method closely resembling JN iteration is implemented as a separate method option in LSODE. It is like JN iteration in that it uses a diagonal approximation **D** to the Jacobian matrix. However, the diagonal elements D_{ii} are, in general, different from J_{ii} and are given by the difference quotient

$$D_{ii} = \frac{f_i(\underline{Y} + \Delta\underline{Y}) - f_i(\underline{Y})}{\Delta Y_i}, \qquad i = 1, \ldots, N, \tag{2.31}$$

where the increment vector $\Delta\underline{Y} = 0.1\beta_0\,\underline{g}(\underline{Y}_n^{[0]})$. If **J** is actually a diagonal matrix, $D_{ii} = J_{ii} + O(\Delta Y_i^2)$, but, in general, D_{ii} effectively lumps together the various elements $\{J_{ij}\}$ in row i of **J**.

2.2.4 Unified Formulation

The different iteration methods can be generalized by the recursive relations

$$\underline{Y}_n^{[m+1]} = \underline{Y}_n^{[m]} + \beta_0 \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right) \tag{2.32}$$

and

$$h_n \dot{\underline{Y}}_n^{[m+1]} = h_n \dot{\underline{Y}}_n^{[m]} + \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right), \tag{2.33}$$

where **P** depends on the iteration method. For functional iteration **P** = **I**, and for NR and JN iterations **P** is given by equation (2.25), where **J** is the appropriate Jacobian matrix, equation (1.5), (2.30), or (2.31).

The combination of equations (2.32) and (2.33) gives

$$\underline{Y}_n^{[m+1]} - \underline{Y}_n^{[m]} = h_n\beta_0\left(\dot{\underline{Y}}_n^{[m+1]} - \dot{\underline{Y}}_n^{[m]}\right), \tag{2.34}$$

which shows that if $\underline{Y}_n^{[m]}$ converges as $m \to \infty$, so does $\dot{\underline{Y}}_n^{[m]}$. Equation (2.32) shows that if $\underline{Y}_n^{[m]}$ converges (to $\underline{Y}_n$) as $m \to \infty$, $\underline{g}(\underline{Y}_n^{[m]}) \to 0$, and therefore we see from equations (2.15) and (2.16), respectively, (1) that the converged solution satisfies equation (2.5) and (2) that $\dot{\underline{Y}}_n^{[m]} \to \underline{f}(\underline{Y}_n) = \underline{f}_n$.

The predictor-corrector methods can be summarized as follows:

Predictor:

$$\left.\begin{aligned} \underline{Y}_n^{[0]} &= \underline{\psi}_n^* \\ h_n\dot{\underline{Y}}_n^{[0]} &= \frac{\underline{Y}_n^{[0]} - \underline{\psi}_n}{\beta_0}. \end{aligned}\right\} \tag{2.35}$$

Corrector:

$$\left.\begin{aligned} \underline{g}\left(\underline{Y}_n^{[m]}\right) &= h_n\underline{f}\left(\underline{Y}_n^{[m]}\right) - h_n\dot{\underline{Y}}_n^{[m]} \\ \underline{Y}_n^{[m+1]} &= \underline{Y}_n^{[m]} + \beta_0\mathbf{P}^{-1}\underline{g}\left(\underline{Y}_n^{[m]}\right) \\ h_n\dot{\underline{Y}}_n^{[m+1]} &= h_n\dot{\underline{Y}}_n^{[m]} + \mathbf{P}^{-1}\underline{g}\left(\underline{Y}_n^{[m]}\right) \end{aligned}\right\} \quad m = 0, 1, \ldots, M-1. \tag{2.36}$$

$$\left.\begin{aligned} \underline{Y}_n &= \underline{Y}_n^{[M]} \\ h_n\dot{\underline{Y}}_n &= h_n\dot{\underline{Y}}_n^{[M]}. \end{aligned}\right\} \tag{2.37}$$

2.3 Matrix Formulation

The implementation of linear multistep methods is aided by a matrix formulation (ref. 21). This formulation, constructed by Gear (ref. 18), is summarized here.

To solve for $\underline{Y}_n$ and $h_n\dot{\underline{Y}}_n$ by using equations (2.35) to (2.37), we need, and therefore must have saved, the $L = q + 1$ column vectors $\underline{Y}_{n-1}$, $h_n\dot{\underline{Y}}_{n-1}$, $h_n\dot{\underline{Y}}_{n-2}, \ldots,$ and $h_n\dot{\underline{Y}}_{n-q}$ for the AM method of order q, or $\underline{Y}_{n-1}, \underline{Y}_{n-2}, \ldots, \underline{Y}_{n-q}$, and $h_n\dot{\underline{Y}}_{n-1}$ for the BDF method of order q. Hence for the AM method of order q we define the $N{\times}L$ history matrix $\mathbf{w}_{n-1}$ at ξ_{n-1} by

$$\mathbf{w}_{n-1} = \left(\underline{Y}_{n-1}, h_n\dot{\underline{Y}}_{n-1}, h_n\dot{\underline{Y}}_{n-2}, \ldots, h_n\dot{\underline{Y}}_{n-q}\right), \tag{2.38a}$$

that is,

$$\mathbf{w}_{n-1} = \begin{pmatrix} Y_{1,n-1} & h_n\dot{Y}_{1,n-1} & h_n\dot{Y}_{1,n-2} & \cdots & h_n\dot{Y}_{1,n-q} \\ Y_{2,n-1} & h_n\dot{Y}_{2,n-1} & h_n\dot{Y}_{2,n-2} & \cdots & h_n\dot{Y}_{2,n-q} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ Y_{N,n-1} & h_n\dot{Y}_{N,n-1} & h_n\dot{Y}_{N,n-2} & \cdots & h_n\dot{Y}_{N,n-q} \end{pmatrix}. \tag{2.39}$$

The updated matrix

$$\mathbf{w}_n = \left(\underline{Y}_n, h_n\dot{\underline{Y}}_n, h_n\dot{\underline{Y}}_{n-1}, \ldots, h_n\dot{\underline{Y}}_{n-q+1}\right) \tag{2.40a}$$

is then constructed at each step ξ_n. The predicted matrix $\mathbf{w}_n^{[0]}$ at ξ_n is given by

$$\mathbf{w}_n^{[0]} = \left(\underline{Y}_n^{[0]}, h_n\dot{\underline{Y}}_n^{[0]}, h_n\dot{\underline{Y}}_{n-1}, \ldots, h_n\dot{\underline{Y}}_{n-q+1}\right). \tag{2.41a}$$

For the BDF method of order q these matrices take the form

$$\mathbf{w}_{n-1} = \left(\underline{Y}_{n-1}, h_n\dot{\underline{Y}}_{n-1}, \underline{Y}_{n-2}, \ldots, \underline{Y}_{n-q}\right), \tag{2.38b}$$

$$\mathbf{w}_n = \left(\underline{Y}_n, h_n\dot{\underline{Y}}_n, \underline{Y}_{n-1}, \ldots, \underline{Y}_{n-q+1}\right), \tag{2.40b}$$

and

$$\mathbf{w}_n^{[0]} = \left(\underline{Y}_n^{[0]}, h_n\dot{\underline{Y}}_n^{[0]}, \underline{Y}_{n-1}, \ldots, \underline{Y}_{n-q+1}\right). \tag{2.41b}$$

The matrix formulations for $\mathbf{w}_n^{[0]}$ and $\mathbf{w}_n$ are derived as follows: Substituting the expression for $\underline{Y}_n^{[0]}$, equation (2.8a) or (2.8b), into that for $h_n\dot{\underline{Y}}_n^{[0]}$, equation (2.35), and then using equation (2.6a) or (2.6b) give

$$h_n\dot{\underline{Y}}_n^{[0]} = \sum_{j=1}^{q-1}\left(\frac{\beta_j^* - \beta_j}{\beta_0}\right) h_n\dot{\underline{Y}}_{n-j} + \frac{\beta_q^*}{\beta_0}\, h_n\dot{\underline{Y}}_{n-q} \tag{2.42a}$$

for the AM method of order q and

$$h_n\dot{\underline{Y}}_n^{[0]} = \sum_{j=1}^{q}\left(\frac{\alpha_j^* - \alpha_j}{\beta_0}\right) \underline{Y}_{n-j} + \frac{\beta_1^*}{\beta_0} h_n\dot{\underline{Y}}_{n-1} \tag{2.42b}$$

for the BDF method of order q. Equations (2.8a) and (2.42a), or (2.8b) and (2.42b), that is, the prediction process, can be rewritten as the matrix equation

$$\mathbf{w}_n^{[0]} = \mathbf{w}_{n-1}\mathbf{B}, \tag{2.43}$$

where the $L \times L$ matrix $\mathbf{B}$ depends on the solution method. For the AM method of order q, it is given by

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & 0 & . & . & . & 0 & 0 \\ \beta_1^* & \dfrac{\beta_1^* - \beta_1}{\beta_0} & 1 & 0 & . & . & . & 0 & 0 \\ \beta_2^* & \dfrac{\beta_2^* - \beta_2}{\beta_0} & 0 & 1 & . & . & . & 0 & 0 \\ . & . & . & . & & & & . & . \\ . & . & . & . & & & & . & . \\ . & . & . & . & & & & 1 & . \\ \beta_{q-1}^* & \dfrac{\beta_{q-1}^* - \beta_{q-1}}{\beta_0} & 0 & 0 & . & . & . & 0 & 1 \\ \beta_q^* & \dfrac{\beta_q^*}{\beta_0} & 0 & 0 & . & . & . & 0 & 0 \end{pmatrix} \tag{2.44a}$$

and for the BDF method of order q,

$$\mathbf{B} = \begin{pmatrix} \alpha_1^* & \dfrac{\alpha_1^* - \alpha_1}{\beta_0} & 1 & 0 & 0 & . & . & . & 0 \\ \beta_1^* & \dfrac{\beta_1^*}{\beta_0} & 0 & 0 & 0 & . & . & . & 0 \\ \alpha_2^* & \dfrac{\alpha_2^* - \alpha_2}{\beta_0} & 0 & 1 & 0 & . & . & . & 0 \\ \alpha_3^* & \dfrac{\alpha_3^* - \alpha_3}{\beta_0} & 0 & 0 & 1 & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & 1 & . \\ \alpha_{q-1}^* & \dfrac{\alpha_{q-1}^* - \alpha_{q-1}}{\beta_0} & 0 & 0 & 0 & . & . & . & 1 \\ \alpha_q^* & \dfrac{\alpha_q^* - \alpha_q}{\beta_0} & 0 & 0 & 0 & . & . & . & 0 \end{pmatrix}. \qquad (2.44b)$$

The corrector equation, equation (2.36), can be expressed in matrix form as

$$\mathbf{w}_n^{[m+1]} = \mathbf{w}_n^{[m]} + \mathbf{P}^{-1}\underline{g}\left(\underline{Y}_n^{[m]}\right)\underline{k}, \qquad (2.45)$$

where $\mathbf{w}_n^{[m]}$, the history matrix on the mth iteration, is given by

$$\mathbf{w}_n^{[m]} = \left(\underline{Y}_n^{[m]}, h_n\dot{\underline{Y}}_n^{[m]}, h_n\dot{\underline{Y}}_{n-1}, \ldots, h_n\dot{\underline{Y}}_{n-q+1}\right) \qquad (2.46a)$$

for the AM method and by

$$\mathbf{w}_n^{[m]} = \left(\underline{\mathrm{Y}}_n^{[m]}, h_n \dot{\underline{\mathrm{Y}}}_n^{[m]}, \underline{\mathrm{Y}}_{n-1}, \ldots, \underline{\mathrm{Y}}_{n-q+1} \right) \tag{2.46b}$$

for the BDF method, $\underline{\ell}$ is the L-dimensional vector

$$\underline{\ell} = \left(\beta_0, 1, 0, \ldots, 0 \right), \tag{2.47}$$

and $\mathbf{P}$ depends on the iteration technique, as described in section 2.2.4.

The matrix formulation of the methods can be summarized as follows:

Predictor:

$$\mathbf{w}_n^{[0]} = \mathbf{w}_{n-1}\mathbf{B}. \tag{2.48}$$

Corrector:

$$\left.\begin{aligned} \underline{\mathrm{g}}\left(\underline{\mathrm{Y}}_n^{[m]}\right) &= h_n \underline{\mathrm{f}}\left(\underline{\mathrm{Y}}_n^{[m]}\right) - h_n \dot{\underline{\mathrm{Y}}}_n^{[m]} \\ \mathbf{w}_n^{[m+1]} &= \mathbf{w}_n^{[m]} + \mathbf{P}^{-1} \underline{\mathrm{g}}\left(\underline{\mathrm{Y}}_n^{[m]}\right) \underline{\ell} \end{aligned}\right\} \quad m = 0, 1, \ldots, M-1. \tag{2.49}$$

$$\mathbf{w}_n = \mathbf{w}_n^{[M]}. \tag{2.50}$$

2.4 Nordsieck's History Matrix

Instead of saving information in the form $\mathbf{w}_{n-1}$, equation (2.38a) or (2.38b), Gear (ref. 18) suggested making a linear transformation and storing the matrix $\mathbf{z}_{n-1}$ given by

$$\mathbf{z}_{n-1} = \mathbf{w}_{n-1}\mathbf{Q}, \tag{2.51}$$

where the LxL transformation matrix $\mathbf{Q}$ is nonsingular. In particular, $\mathbf{Q}$ is chosen such that the matrix representation suggested by Nordsieck (ref. 33) is obtained:

$$\mathbf{z}_{n-1} = \left(\underline{\mathrm{Y}}_{n-1}, h_n \dot{\underline{\mathrm{Y}}}_{n-1}, \frac{h_n^2}{2!} \ddot{\underline{\mathrm{Y}}}_{n-1}, \ldots, \frac{h_n^q}{q!} \underline{\mathrm{Y}}_{n-1}^{(q)} \right), \tag{2.52}$$

that is, the $N{\times}L$ matrix $\mathbf{z}_{n-1}$ is given by

$$\mathbf{z}_{n-1} = \begin{pmatrix} Y_{1,n-1} & h_n\dot{Y}_{1,n-1} & \cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot & \frac{h_n^q}{q!}Y_{1,n-1}^{(q)} \\ Y_{2,n-1} & h_n\dot{Y}_{2,n-1} & \cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot & \frac{h_n^q}{q!}Y_{2,n-1}^{(q)} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ Y_{N,n-1} & h_n\dot{Y}_{N,n-1} & \cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot & \frac{h_n^q}{q!}Y_{N,n-1}^{(q)} \end{pmatrix}. \tag{2.53}$$

In equation (2.53), $Y_{i,n-1}^{(j)}$ is the jth derivative of the approximating polynomial for $Y_{i,n-1}$. Because scaled derivatives $h_n^j\underline{Y}_{n-1}^{(j)}/j!$ are used, $\mathbf{Q}$ is independent of the step size. However, $\mathbf{Q}$ depends on the solution method. The N rows of $\mathbf{z}_{n-1}$ are numbered from 1 to N, so that the ith row ($i = 1,...,N$) contains the $q + 1$ scaled derivatives of the ith component, $Y_{i,n-1}$, of $\underline{Y}_{n-1}$. The $q + 1$ columns are, however, numbered from 0 to q, so that the column number corresponds to the order of the scaled derivative stored in that column. Thus the jth column ($j = 0,1,...,q$), which we denote by the vector $\underline{z}_{n-1}(j)$, contains the vector $h_n^j\underline{Y}_{n-1}^{(j)}/j!$. The Nordsieck matrix formulation of the method is referred to as the "normal form of the method" (ref. 10).

Applying the appropriate transformation matrix $\mathbf{Q}$ to the predictor equation, equation (2.48), gives

$$\mathbf{z}_n^{[0]} = \mathbf{w}_n^{[0]}\mathbf{Q} = \mathbf{w}_{n-1}\mathbf{B}\mathbf{Q} = \mathbf{z}_{n-1}\mathbf{Q}^{-1}\mathbf{B}\mathbf{Q} = \mathbf{z}_{n-1}\mathbf{A}, \tag{2.54}$$

where

$$\mathbf{z}_n^{[0]} = \left(\underline{Y}_n^{[0]}, h_n\dot{\underline{Y}}_n^{[0]}, \frac{h_n^2}{2!}\ddot{\underline{Y}}_n^{[0]}, ..., \frac{h_n^q}{q!}\underline{Y}_n^{[0](q)}\right), \tag{2.55}$$

is the predicted $N{\times}L$ Nordsieck history matrix at ξ_n and

$$\mathbf{A} = \mathbf{Q}^{-1}\mathbf{B}\mathbf{Q}. \tag{2.56}$$

The $L{\times}L$ prediction matrix **A** provides a qth-order approximation to $\mathbf{z}_n^{[0]}$ in terms of $\mathbf{z}_{n-1}$ and is therefore the lower-triangular Pascal triangle matrix (ref. 10), with element A_{ij} given by

$$A_{ij} = \begin{cases} 0, & i < j \\ \binom{i}{j}, & i \geq j \end{cases} \qquad i,j = 0,1,\ldots,q, \tag{2.57}$$

where $\binom{i}{j}$ is the binomial coefficient, defined as

$$\binom{i}{j} = \frac{i!}{j!(i-j)!}. \tag{2.58}$$

Hence

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & \cdots & \cdot & \cdot & \cdot & \cdot \\ 1 & 2 & 1 & & \cdots & \cdot & & & \\ 1 & 3 & 3 & 1 & 0 & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & & & & \\ \cdot & & & & \cdot & & \cdot & & \\ 1 & q-2 & \frac{(q-2)(q-3)}{2!} & \frac{(q-2)(q-3)(q-4)}{3!} & \cdot & \cdots & 1 & \cdot & \\ 1 & q-1 & \frac{(q-1)(q-2)}{2!} & \frac{(q-1)(q-2)(q-3)}{3!} & \cdots & \cdots & (q-1) & 1 & \cdot \\ 1 & q & \frac{q(q-1)}{2!} & \frac{q(q-1)(q-2)}{3!} & \cdots & & \frac{q(q-1)}{2!} & q & 1 \end{pmatrix} \tag{2.59}$$

The principal advantage of using the Nordsieck history matrix is that the matrix multiplication implied by equation (2.54) can be carried out solely by repeated additions, as shown by Gear (ref. 10). Hence computer multiplications are avoided, resulting in considerable savings of computational effort for large problems. Also **A** need not be stored and $\mathbf{z}_n^{[0]}$ overwrites $\mathbf{z}_{n-1}$, thereby reducing memory requirements.

Because

$$\binom{i+1}{j+1} = \binom{i}{j+1} + \binom{i}{j} \tag{2.60}$$

and $A_{ii} = A_{i0} = 1$ for all i, the product **zA** is computed as follows (refs. 10 and 15):

```
For $k = 0, 1, ..., q-1$, do:
    For $j = q, q-1, ..., k+1$, do:                    (2.61)
        $z_{i,j-1} \leftarrow z_{i,j} + z_{i,j-1}, \quad i = 1, ..., N.$
```

In this equation the subscripts n and $n-1$ have been dropped because the **z** values do not indicate any one value of ξ but represent a continuous replacement process. At the start of the calculation procedure given by equation (2.61), $\mathbf{z} = \mathbf{z}_{n-1}$; and at the end $\mathbf{z} = \mathbf{z}_n^{[0]}$. The arrow "$\leftarrow$" denotes the replacement operator, which means overwriting the contents of a computer storage location. For example,

$$z_{i,3} \leftarrow z_{i,4} + z_{i,3}$$

means that $z_{i,4}$ is added to $z_{i,3}$ and the result replaces the contents of the location $z_{i,3}$. The total number of additions required in equation (2.61) is $Nq(q + 1)/2$. The predictor step is a Taylor series expansion about the previous point ξ_{n-1} and is independent of both the integration method and the ODE.

Another important advantage of using Nordsieck's formulation is that it makes changing step size easy. For example, if at ξ_n the step size is changed from h_n to rh_n, the new history matrix is obtained from

$$\mathbf{z}_n \leftarrow \mathbf{z}_n\mathbf{C}, \tag{2.62}$$

where the $L \times L$ diagonal matrix **C** is given by

$$\mathbf{C} = \begin{pmatrix} 1 & & & & & & 0 \\ & r & & & & & \\ & & r^2 & & & & \\ & & & \cdot & & & \\ & & & & \cdot & & \\ & & & & & \cdot & \\ & 0 & & & & & r^q \end{pmatrix}. \tag{2.63}$$

The rescaling can be done by multiplications alone, as follows:

$$\begin{aligned}&R = 1\\ &\text{For } j = 1,\dots,q, \text{ do:}\\ &\quad\begin{cases} R \leftarrow rR \\ z_{i,j} \leftarrow z_{i,j}R, \; i = 1,\dots,N. \end{cases}\end{aligned} \tag{2.64}$$

The corrector equation corresponding to equation (2.49) is given by

$$\mathbf{z}_n^{[m+1]} = \mathbf{w}_n^{[m+1]}\,\mathbf{Q} = \mathbf{w}_n^{[m]}\,\mathbf{Q} + \mathbf{P}^{-1}\underline{g}\left(\underline{Y}_n^{[m]}\right)\underline{k}\mathbf{Q} = \mathbf{z}_n^{[m]} + \mathbf{P}^{-1}\underline{g}\left(\underline{Y}_n^{[m]}\right)\underline{\ell}, \tag{2.65}$$

where $\mathbf{z}_n^{[m]}$, the Nordsieck history matrix on the mth iteration, is given by

$$\mathbf{z}_n^{[m]} = \left(\underline{Y}_n^{[m]}, h_n\dot{\underline{Y}}_n^{[m]}, \frac{h_n^2}{2!}\ddot{\underline{Y}}_n^{[m]}, \dots, \frac{h_n^q}{q!}\underline{Y}_n^{[m](q)}\right) \tag{2.66}$$

and

$$\underline{\ell} = \underline{k}\mathbf{Q} \tag{2.67}$$

is an L-dimensional vector

$$\underline{\ell} = \left(\ell_0, \ell_1, \dots, \ell_q\right). \tag{2.68}$$

For the two solution methods used in LSODE the values of $\underline{\ell}$ are derived in references 21 and 22 and reproduced in tables 2.1 and 2.2. Methods expressed in the form of equations (2.54) and (2.65) are better described as multivalue or L-value methods than multistep methods (ref. 10) because it is the number L of values saved from step to step that is significant and not the number of steps involved.

The two matrix formulations described here are related by the transformation equations (2.51), (2.54), and (2.65) and are therefore said to be equivalent (ref. 10). The equivalence means that if the step $[\xi_{n-1},\xi_n]$ is taken by the two methods with equivalent past values $\mathbf{w}_{n-1}$ and $\mathbf{z}_{n-1}$, that is, related by equation (2.51) through $\mathbf{Q}$, then the resulting solutions $\mathbf{w}_n$ and $\mathbf{z}_n$ will also be related by equation (2.51) through $\mathbf{Q}$, apart from roundoff errors (ref. 21). The transformation does not affect the stability properties or the accuracy of the

TABLE 2.1.—METHOD COEFFICIENTS FOR ADAMS-MOULTON METHOD IN NORMAL FORM OF ORDERS 1 TO 12

q	ℓ_0	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6	ℓ_7	ℓ_8	ℓ_9	ℓ_{10}	ℓ_{11}	ℓ_{12}
1	1	1											
2	$\frac{1}{2}$	1	$\frac{1}{2}$										
3	$\frac{5}{12}$	1	$\frac{3}{4}$	$\frac{1}{6}$									
4	$\frac{3}{8}$	1	$\frac{11}{12}$	$\frac{1}{3}$	$\frac{1}{24}$								
5	$\frac{251}{720}$	1	$\frac{25}{24}$	$\frac{35}{72}$	$\frac{5}{48}$	$\frac{1}{120}$							
6	$\frac{95}{288}$	1	$\frac{137}{120}$	$\frac{5}{8}$	$\frac{17}{96}$	$\frac{1}{40}$	$\frac{1}{720}$						
7	$\frac{19087}{60480}$	1	$\frac{49}{40}$	$\frac{203}{270}$	$\frac{49}{192}$	$\frac{7}{144}$	$\frac{7}{1440}$	$\frac{1}{5040}$					
8	$\frac{5257}{17280}$	1	$\frac{363}{280}$	$\frac{469}{540}$	$\frac{967}{2880}$	$\frac{7}{90}$	$\frac{23}{2160}$	$\frac{1}{1260}$	$\frac{1}{40320}$				
9	$\frac{1070017}{3628800}$	1	$\frac{761}{560}$	$\frac{29531}{30240}$	$\frac{267}{640}$	$\frac{1069}{9600}$	$\frac{3}{160}$	$\frac{13}{6720}$	$\frac{1}{8960}$	$\frac{1}{362880}$			
10	$\frac{25713}{89600}$	1	$\frac{7129}{5040}$	$\frac{6515}{6048}$	$\frac{4523}{9072}$	$\frac{19}{128}$	$\frac{3013}{103680}$	$\frac{5}{1344}$	$\frac{29}{96768}$	$\frac{1}{72576}$	$\frac{1}{3628800}$		
11	$\frac{26842253}{95800320}$	1	$\frac{7381}{5040}$	$\frac{177133}{151200}$	$\frac{84095}{145152}$	$\frac{341693}{1814400}$	$\frac{8591}{207360}$	$\frac{7513}{1209600}$	$\frac{121}{193536}$	$\frac{11}{272160}$	$\frac{11}{7257600}$	$\frac{1}{39916800}$	
12	$\frac{4777223}{17418240}$	1	$\frac{83711}{55440}$	$\frac{190553}{151200}$	$\frac{341747}{518400}$	$\frac{139381}{604800}$	$\frac{242537}{4354560}$	$\frac{1903}{201600}$	$\frac{10831}{9676800}$	$\frac{11}{120960}$	$\frac{1}{207360}$	$\frac{1}{6652800}$	$\frac{1}{479001600}$

TABLE 2.2.—METHOD COEFFICIENTS FOR BACKWARD DIFFERENTIATION FORMULA METHOD IN NORMAL FORM OF ORDERS 1 TO 6

q	ℓ_0	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
1	1	1					
2	$\frac{2}{3}$	$\frac{3}{3}$	$\frac{1}{3}$				
3	$\frac{6}{11}$	$\frac{11}{11}$	$\frac{6}{11}$	$\frac{1}{11}$			
4	$\frac{24}{50}$	$\frac{50}{50}$	$\frac{35}{50}$	$\frac{10}{50}$	$\frac{1}{50}$		
5	$\frac{120}{274}$	$\frac{274}{274}$	$\frac{225}{274}$	$\frac{85}{274}$	$\frac{15}{274}$	$\frac{1}{274}$	
6	$\frac{720}{1764}$	$\frac{1764}{1764}$	$\frac{1624}{1764}$	$\frac{735}{1764}$	$\frac{175}{1764}$	$\frac{21}{1764}$	$\frac{1}{1764}$

method, but roundoff properties and computational effort depend on the representation used, as discussed by Gear (ref. 10).

The first two columns of $\mathbf{z}_n$ and $\mathbf{w}_n$ are identical (see eqs. (2.38a), (2.38b), and (2.52)), and so $\ell_0 = \beta_0$ and $\ell_1 = 1$. For the same reason the corrector iteration procedures for $\underline{Y}_n$ and $h_n\dot{\underline{Y}}_n$ remain unchanged (see eqs. (2.45), (2.47), and (2.65)). However, to facilitate estimation of the local truncation error, a different iteration procedure than that given by equation (2.65) is used. To derive the new formulation, $\mathbf{z}_n^{[m+1]}$ is written as

$$\mathbf{z}_n^{[m+1]} = \mathbf{z}_n^{[m+1]} - \mathbf{z}_n^{[m]} + \mathbf{z}_n^{[m]} - \mathbf{z}_n^{[m-1]} + \ldots + \mathbf{z}_n^{[1]} - \mathbf{z}_n^{[0]} + \mathbf{z}_n^{[0]}$$

or

$$\mathbf{z}_n^{[m+1]} = \mathbf{z}_n^{[0]} + \sum_{j=0}^{m} \left(\mathbf{z}_n^{[j+1]} - \mathbf{z}_n^{[j]} \right). \tag{2.69}$$

Substituting the difference $\mathbf{z}_n^{[j+1]} - \mathbf{z}_n^{[j]}$ obtained from equation (2.65) into equation (2.69) produces

$$\mathbf{z}_n^{[m+1]} = \mathbf{z}_n^{[0]} + \sum_{j=0}^{m} \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[j]} \right) \underline{\ell} = \mathbf{z}_n^{[0]} + \underline{e}_n^{[m+1]} \underline{\ell}, \tag{2.70}$$

where $\underline{e}_n^{[m+1]}$ is defined as

$$\underline{e}_n^{[m+1]} = \sum_{j=0}^{m} \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[j]}\right). \tag{2.71}$$

It is clear from this equation that

$$\underline{e}_n^{[m+1]} = \underline{e}_n^{[m]} + \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right). \tag{2.72}$$

Equation (2.70) can be used to rewrite $\underline{g}(\underline{Y}_n^{[m]})$, equation (2.16), as follows:

$$\underline{g}\left(\underline{Y}_n^{[m]}\right) = h_n \underline{f}\left(\underline{Y}_n^{[m]}\right) - h_n \dot{\underline{Y}}_n^{[0]} - \underline{e}_n^{[m]}, \tag{2.73}$$

because $\ell_1 = 1$.

Finally, because only the first two columns of $\mathbf{z}_n$ enter into the solution of equation (2.5), the successive corrections can be accumulated and applied to the remaining columns of $\mathbf{z}_n$ after convergence. Clearly, not updating all columns of the Nordsieck history matrix after each iteration results in savings of computational effort, especially when a high-order method is used and/or the number of ODE's is large. For additional savings of computer time the history matrix is updated only if both (1) the iteration converges and (2) the converged solution satisfies accuracy requirements.

The predictor-corrector formulation utilized in LSODE can be summarized as follows:

Predictor:

$$\left.\begin{aligned} \mathbf{z}_n^{[0]} &= \mathbf{z}_{n-1}\mathbf{A} \\ \underline{e}_n^{[0]} &= 0. \end{aligned}\right\} \tag{2.74}$$

Corrector:

$$\left.\begin{aligned} \underline{g}\left(\underline{Y}_n^{[m]}\right) &= h_n \underline{f}\left(\underline{Y}_n^{[m]}\right) - h_n \dot{\underline{Y}}_n^{[0]} - \underline{e}_n^{[m]} \\ \underline{e}_n^{[m+1]} &= \underline{e}_n^{[m]} + \mathbf{P}^{-1} \underline{g}\left(\underline{Y}_n^{[m]}\right) \\ \underline{Y}_n^{[m+1]} &= \underline{Y}_n^{[0]} + \ell_0 \underline{e}_n^{[m+1]} \end{aligned}\right\} \quad m = 0, 1, \ldots, M-1. \tag{2.75}$$

$$\left.\begin{aligned} \underline{e}_n &= \underline{e}_n^{[M]} \\ \mathbf{z}_n &= \mathbf{z}_n^{[0]} + \underline{e}_n \underline{\ell}. \end{aligned}\right\} \qquad (2.76)$$

2.5 Local Truncation Error Estimate and Control[1]

The local truncation error is defined to be the amount by which the exact solution $\underline{y}(\xi)$ to the ODE system fails to satisfy the difference equation of the numerical method (refs. 4, 10, 12, and 26). That is, for the linear multistep methods, equation (2.1), the local truncation error vector $\underline{d}_n$ at ξ_n is the residual in the difference formula when the approximations $\{\underline{Y}_j\}$ and $\{\underline{f}_j\}$ are replaced by the exact solution and its derivative.[2] In LSODE, however, the basic multistep formula is normalized by dividing it by

$$\sum_{j=0}^{K_2} \beta_j$$

[1]Although the corrector convergence test is performed before the local truncation error test (which is done only if the iteration converges), we discuss the accuracy test first because the convergence test is based on it.

[2]As discussed in chapter 1, another commonly used definition for the local truncation error is that it is the error incurred by the numerical method in advancing the approximate solution by a single step assuming exact past values and no roundoff errors (refs. 12, 13, and 21). That is, $\underline{d}_n$ is the difference between the numerical approximation $\underline{Y}_n^*$ obtained by using exact past values (i.e., $\{\underline{y}(\xi_{n-j})\}$ and $\{\dot{\underline{y}}(\xi_{n-j})\}$) and the exact solution $\underline{y}(\xi_n)$:

$$\underline{d}_n = \underline{Y}_n^* - \underline{y}(\xi_n), \qquad (2.77)$$

where, for example,

$$\underline{Y}_n^* = \sum_{j=1}^{q} \alpha_j \underline{y}(\xi_{n-j}) + h_n \beta_0 \underline{f}\left(\underline{Y}_n^*\right) \qquad (2.78)$$

for the BDF method of order q. For an explicit method the local truncation error given by equation (2.77) and that obtained by using the definition given in the text above (i.e., the residual of eq. (2.1)) have the same magnitude. However, for an implicit method the two quantities are only approximately proportional to one another (ref. 4), although they agree asymptotically in the limit of small step size.

for reasons given by Henrici (ref. 29) and Gear (ref. 10); however, see Lambert (ref. 4). For example, the BDF method of order q, equation (2.4), can be expressed in this form as

$$0 = \sum_{j=0}^{q} \left(\frac{\alpha_j}{\beta_0} \right) \underline{Y}_{n-j} + h_n \underline{f}_n, \tag{2.79}$$

where $\alpha_0 = -1$. The local truncation error for this method is then given by

$$\underline{d}_n = \sum_{j=0}^{q} \left(\frac{\alpha_j}{\beta_0} \right) \underline{y}\left(\xi_{n-j}\right) + h_n \underline{\dot{y}}\left(\xi_n\right), \tag{2.80}$$

where $\underline{d}_n$ consists of N components

$$\underline{d}_n = \left(d_{1,n}, \ldots, d_{N,n}\right)^T. \tag{2.81}$$

If we assume that each y_i $(i = 1,\ldots,N)$ possesses derivatives of arbitrarily high order, each $y_i(\xi_{n-j})$ $(i = 1,\ldots,N;\ j = 1,\ldots,q)$ in equation (2.80) can be expanded in a Taylor series about ξ_n. Upon collecting terms the resulting expression for $\underline{d}_n$ can be stated compactly as

$$\underline{d}_n = \sum_{k=0}^{\infty} C_k h_n^k\, \underline{y}^{(k)}\left(\xi_n\right), \tag{2.82}$$

where the $\{C_k\}$ are constants (e.g., ref. 10). A method is said to be of order q if $C_0 = C_1 = \ldots = C_q = 0$, and $C_{q+1} \neq 0$. The local truncation error is then given by

$$\underline{d}_n = C_{q+1} h_n^{q+1} \underline{y}^{(q+1)}\left(\xi_n\right) + O\left(h_n^{q+2}\right), \tag{2.83}$$

where the terms C_{q+1} and $C_{q+1} h_n^{q+1}\, \underline{y}^{(q+1)}(\xi_n)$ are, respectively, called the error constant and the principal local truncation error (ref. 4). In particular, for the BDF method of order q in the normalized form given by equation (2.79) (refs. 22 and 29)

$$C_{q+1} = \frac{1}{q+1}. \tag{2.84a}$$

For the implicit Adams method of order q in normalized form (ref. 22)

$$C_{q+1} = \left|\ell_0(q+1) - \ell_0(q)\right|, \tag{2.84b}$$

where $\ell_0(q)$ and $\ell_0(q+1)$ are, respectively, the zeroth component of the coefficient vectors for the AM method in normalized form of orders q and $(q+1)$.

The $(q+1)$th derivative at ξ_n, $\underline{y}^{(q+1)}(\xi_n)$, is estimated as follows: As discussed in section 2.4, at each step the solution method updates the Nordsieck history matrix $\mathbf{z}_n$:

$$\mathbf{z}_n = \left(\underline{Y}_n, h_n\dot{\underline{Y}}_n, \frac{h_n^2}{2!}\ddot{\underline{Y}}_n, \ldots, \frac{h_n^q}{q!}\underline{Y}_n^{(q)}\right). \tag{2.85}$$

For either method of order q the last column of $\mathbf{z}_n$, $\underline{z}_n(q)$, contains the vector $h_n^q\underline{Y}_n^{(q)}/q!$, which is the approximation to $h_n^q\underline{y}^{(q)}(\xi_n)/q!$. Now the prediction step being a Taylor series method of order q does not alter the last column of $\mathbf{z}_{n-1}$, namely the vector $h_n^q\underline{Y}_{n-1}^{(q)}/q!$. Hence the last column of $\mathbf{z}_n^{[0]}$, $\underline{z}_n^{[0]}(q)$, contains the vector $h_n^q\underline{Y}_{n-1}^{(q)}/q!$. The difference, $\underline{z}_n^{(q)} - \underline{z}_n^{[0]}(q)$, is given by

$$\underline{z}_n(q) - \underline{z}_n^{[0]}(q) = \frac{h_n^q}{q!}\underline{Y}_n^{(q)} - \frac{h_n^q}{q!}\underline{Y}_{n-1}^{(q)} = \frac{h_n^{q+1}}{q!}\underline{Y}_n^{(q+1)} + O\left(h_n^{q+2}\right) \tag{2.86}$$

by using the mean value theorem for derivatives. However, equation (2.76) gives the following expression for $\underline{z}_n(q) - \underline{z}_n^{[0]}(q)$:

$$\underline{z}_n(q) - \underline{z}_n^{[0]}(q) = \ell_q\underline{e}_n. \tag{2.87}$$

Equating equations (2.86) and (2.87) gives the following approximation for $h_n^{q+1}\underline{Y}_n^{(q+1)}$ if higher-order terms are neglected:

$$h_n^{q+1}\,\underline{Y}_n^{(q+1)} \cong q!\,\ell_q\underline{e}_n. \tag{2.88}$$

Substituting this equation into equation (2.83) and neglecting higher-order terms give the following estimate for $\underline{d}_n$:

$$\underline{d}_n = C_{q+1}q!\,\ell_q\underline{e}_n. \tag{2.89}$$

In order to provide for user control of the local truncation error, it is normalized by the error weight vector $\underline{\mathrm{EWT}}_n$, with element $\mathrm{EWT}_{i,n}$ defined by

$$\mathrm{EWT}_{i,n} = \mathrm{RTOL}_i \left| \mathrm{Y}_{i,n-1} \right| + \mathrm{ATOL}_i, \tag{2.90}$$

where the user-supplied local relative (RTOL_i) and absolute (ATOL_i) error tolerances for the *i*th solution component are discussed in chapter 4. The solution $\underline{\mathrm{Y}}_n$ is accepted as sufficiently accurate if the following inequality is satisfied:

$$\left\| \underline{d}_n \right\| \equiv \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{d_{i,n}}{\mathrm{EWT}_{i,n}} \right)^2} \overset{?}{\leq} 1, \tag{2.91}$$

where $\|\bullet\|$ denotes the weighted root-mean-square (rms) norm, which is used for reasons discussed by Hindmarsh (ref. 15). Equation (2.91) can be rewritten as

$$\left\| \underline{e}_n \right\| \equiv \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{e_{i,n}}{\mathrm{EWT}_{i,n}} \right)^2} \overset{?}{\leq} \frac{1}{C_{q+1} q! \ell_q}, \tag{2.92}$$

by using equation (2.89). If we define the test coefficient $\tau(q,q)$ as

$$\tau(q,q) = \frac{1}{C_{q+1} q! \ell_q}, \tag{2.93}$$

the accuracy test, equation (2.92), becomes

$$\left\| \underline{e}_n \right\| \overset{?}{\leq} \tau(q,q). \tag{2.94}$$

If we further define the quantity D_q by

$$D_q = \frac{\left\| \underline{e}_n \right\|}{\tau(q,q)}, \tag{2.95}$$

the accuracy test reduces to

$$D_q \overset{?}{\leq} 1. \tag{2.96}$$

The reason for using two variables in the definition for τ will become apparent when we discuss step size and method order selection in section 2.7.

2.6 Corrector Convergence Test and Control

The test for corrector convergence is independent of both the integration method and the iteration technique and is determined by the magnitude of the successive differences $h_n\dot{\underline{Y}}_n^{[m]} - h_n\dot{\underline{Y}}_n^{[m-1]}$. To provide for user control of the convergence process, the difference $h_n\dot{\underline{Y}}_n^{[m]} - h_n\dot{\underline{Y}}_n^{[m-1]}$ is normalized by the error weight vector $\underline{\mathrm{EWT}}_n$, equation (2.90). Now, equation (2.33) provides the following expression for $h_n\dot{\underline{Y}}_n^{[m]} - h_n\dot{\underline{Y}}_n^{[m-1]}$:

$$h_n\dot{\underline{Y}}_n^{[m]} - h_n\dot{\underline{Y}}_n^{[m-1]} = \underline{\delta}_n^{[m]}, \tag{2.97}$$

where we have replaced $\mathbf{P}^{-1}\,\underline{g}(\underline{Y}_n^{[m-1]})$ by $\underline{\delta}_n^{[m]}$. Now, because

$$\underline{e}_n^{[m+1]} = \sum_{j=0}^{m} \underline{\delta}_n^{[j]}$$

(see eq. (2.71)) and the test on $|\underline{e}_n|$ is $|\underline{e}_n| \overset{?}{\leq} \tau(q,q)$, equation (2.94), the following test for convergence

$$\varepsilon_m \equiv \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\delta_{i,n}^{[m]}}{\mathrm{EWT}_{i,n}}\right)^2} \overset{?}{\leq} \frac{\tau(q,q)}{2(q+2)} \tag{2.98}$$

is consistent with the local truncation error test. The empirical factor $2(q + 2)$ in equation (2.98) guarantees that the implicit equation (2.5) is solved to greater accuracy than that required of the numerical solution (refs. 22 and 25).

To increase computational efficiency, especially when the iteration is clearly not converging, LSODE uses the following convergence test instead of equation (2.98):

$$\varepsilon'_m \overset{?}{\leq} \frac{\tau(q,q)}{2(q+2)}. \tag{2.99}$$

The quantity ε'_m is related to ε_m by

$$\varepsilon'_m = \varepsilon_m \quad \min(1, 1.5c'_m), \tag{2.100}$$

where

$$c'_m = \max(0.2c_{m-1}, c_m) \tag{2.101}$$

and

$$c_m = \varepsilon_m / \varepsilon_{m-1} \tag{2.102}$$

is the estimated convergence rate (refs. 22 and 25). Clearly at least two iterations are required before c_m can be computed. For the first iteration c'_m is set equal to the last value of c_m from the previous step. For the first iteration of the very first step and, in the case of NR or JN iteration, after every update of the Jacobian matrix, c'_m is set equal to 0.7. Equation (2.100) assumes that the iteration converges linearly, that is, $\lim_{m\to\infty} (\varepsilon_{m+1}/\varepsilon_m)$ = finite constant c, and essentially anticipates the magnitude of ε_m one iteration in advance (ref. 15). Equation (2.101) shows that the convergence rate of the latest iteration is given much more weight than that of the previous iteration. The rationale for this decision is discussed by Shampine (ref. 25), who examined various practical aspects of implementing implicit methods.

2.7 Step Size and Method Order Selection and Change

Periodically the code attempts to change the step size and/or the method order to minimize computational work while maintaining prescribed accuracy. To minimize complications associated with method order and step size selection, the new order q' is restricted to the values $q - 1$, q, and $q + 1$, where q is the current order. For each q' the step size $h'(q')$ that will satisfy exactly the local error bound is obtained by assuming that the highest derivative remains constant. The method order that produces the largest h' is used on the next step, along with the corresponding h', provided that the h' satisfies certain restrictions described in chapter 3.

For the case $q' = q$, $h'(q)$ is computed by setting $D_q(h')$ (= value of D_q for step size h') = 1 (see eq. (2.96)), so that the local accuracy requirement is satisfied exactly. Then because $\underline{d}_n$ varies as h_n^{q+1} (see eq. (2.83)), we get

$$\frac{D_q}{1} = \left(\frac{h_n}{h'(q)}\right)^{q+1}$$

or

$$r_{\text{same}} \equiv \frac{h'(q)}{h_n} = \left(\frac{1}{D_q}\right)^{\frac{1}{q+1}}, \tag{2.103}$$

where r is the ratio of the step size to be attempted on the next step to its current value. The subscript "same" indicates that the same order used on the current step is to be attempted on the next step.

For the case $q' = q - 1$, $\underline{d}_n(q-1)$ is of order q, where the variable $q-1$ indicates the method order for which the local truncation error is to be estimated, and

$$\underline{d}_n(q-1) = C_q h_n^q \underline{y}^{(q)}(\xi_n), \tag{2.104}$$

where $C_q = |\ell_0(q) - \ell_0(q-1)|$ for the AM method and $1/q$ for the BDF method (refs. 22 and 29). Now, the last column of $\mathbf{z}_n$, $\underline{z}_n(q)$, contains the vector $h_n^q \underline{Y}_n^{(q)}/q!$ (see eq. (2.85)), and so $\underline{d}_n(q-1)$ is easily calculated. On using the rms norm, equation (2.91), the error test for $q' = q - 1$ becomes

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{C_q h_n^q Y_{i,n}^{(q)}}{\mathrm{EWT}_{i,n}}\right)^2} \overset{?}{\leq} 1. \tag{2.105}$$

If we define the test coefficient $\tau(q, q-1)$ as $1/C_q q!$, equation (2.105) can be written as

$$D_{q-1} \equiv \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{h_n^q}{q!}Y_{i,n}^{(q)}}{\mathrm{EWT}_{i,n}}\right)^2}}{\tau(q, q-1)} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{z_{i,n}(q)}{\mathrm{EWT}_{i,n}}\right)^2}}{\tau(q, q-1)} \overset{?}{\leq} 1, \tag{2.106}$$

where $z_{i,n}(q)$ is the ith element of $\underline{z}_n(q)$. The first variable in the definition for τ gives the method order used on the current step. The second variable indicates the method order for which the local truncation error is to be estimated.

The step size $h'(q-1)$ to be attempted on the next step, if the order is reduced to $q-1$, is obtained by using exactly the same procedure that was utilized for the case $q' = q$, that is, by setting $D_{q-1}(h') = 1$. Because $\underline{d}_n(q-1)$ varies as h_n^q, the resulting step size ratio r_{down} is given by

$$r_{\text{down}} \equiv \frac{h'(q-1)}{h_n} = \left(\frac{1}{D_{q-1}}\right)^{\frac{1}{q}}. \tag{2.107}$$

The subscript "down" indicates that the order is to be reduced by 1.

For the case $q' = q + 1$ the local truncation error $\underline{d}_n(q+1)$ is of order $q+2$ and is given by

$$\underline{d}_n(q+1) = C_{q+2} h_n^{q+2} \underline{y}^{(q+2)}(\xi_n), \tag{2.108}$$

where $C_{q+2} = |\ell_0(q+2) - \ell_0(q+1)|$ for the AM method and $1/(q+2)$ for the BDF method (refs. 22 and 29). This case is more difficult than the previous two cases because equation (2.108) involves the derivative of order $q+2$. The derivative $\underline{y}^{(q+2)}(\xi_n)$ is estimated as follows. Equation (2.88) shows that the vector $\ell_q \underline{e}_n$ is approximately proportional to $h_n^{q+1}\underline{Y}_n^{(q+1)}/q!$. We difference the quantity $\ell_q \underline{e}$ over the last two steps and use the mean value theorem for derivatives to get

$$\ell_q \nabla \underline{e}_n \equiv \ell_q \underline{e}_n - \ell_q \underline{e}_{n-1} \cong \frac{h_n^{q+1}}{q!} \underline{Y}_n^{(q+1)} - \frac{h_n^{q+1}}{q!} \underline{Y}_{n-1}^{(q+1)}$$

$$\cong \frac{h_n^{q+2}}{q!} \underline{Y}_n^{(q+2)} + O\left(h_n^{q+3}\right). \tag{2.109}$$

Hence the error test for $q' = q + 1$ becomes

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{C_{q+2}q!\ell_q \nabla e_{i,n}}{\text{EWT}_{i,n}}\right)^2} \overset{?}{\leq} 1, \tag{2.110}$$

where we have again used the rms norm and $\nabla e_{i,n}$ is the ith component of $\nabla \underline{e}_n$. If we define the test coefficient $\tau(q, q+1)$ as $1/(C_{q+2} q! \ell_q)$, the error test, equation (2.110), can be rewritten as

$$D_{q+1} \equiv \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\nabla e_{i,n}}{\text{EWT}_{i,n}}\right)^2}}{\tau(q, q+1)} \overset{?}{\leq} 1. \tag{2.111}$$

To solve for $h'(q+1)$, we use the same procedure as for $h'(q)$ and $h'(q-1)$. The resulting ratio r_{up} is given by

$$r_{\mathrm{up}} \equiv \frac{h'(q+1)}{h_n} = \left(\frac{1}{D_{q+1}}\right)^{\frac{1}{q+2}}. \tag{2.112}$$

The subscript "up" indicates that the order is to be increased by 1.

After a suitable value for the step size ratio r has been computed, the step size h' to be attempted next is calculated:

$$h' = rh_n. \tag{2.113}$$

If the step size and/or the method order is changed, the Nordsieck history matrix has to be modified. For the case $q' = q$ and $h' \neq h_n$, the $q + 1$ columns of $\mathbf{z}_n$ are scaled, as described in section 2.4 (see eqs. (2.62) to (2.64)). For the case $q' = q - 1$ and $h' \neq h_n$, the same scaling is performed on the first q columns; the last column of the old $\mathbf{z}_n$ is ignored because it is not needed on subsequent steps.

If $q' = q + 1$, $\mathbf{z}_n$ must be augmented by a column containing the vector $(h')^{q+1}\underline{Y}_n^{(q+1)}/(q+1)!$. The column addition is done in two stages. First, by using equation (2.88) we derive the following expression for $h_n^{q+1}\underline{Y}_n^{(q+1)}/(q+1)!$:

$$\frac{h_n^{q+1}}{(q+1)!}\underline{Y}_n^{(q+1)} \cong \frac{q!\,\ell_q \underline{e}_n}{(q+1)!} = \frac{\ell_q \underline{e}_n}{q+1} \tag{2.114}$$

and the new column, $\underline{z}_n(q+1)$, is given by

$$\underline{z}_n(q+1) = \frac{\ell_q \underline{e}_n}{q+1}. \tag{2.115}$$

Second, in order to account for any change in the step size, all $q + 2$ columns of $\mathbf{z}_n$ are rescaled as before.

Another factor that must be considered if the step size and/or method order is changed is that the iteration matrix $\mathbf{P}$, equation (2.25), may be altered even if the Jacobian matrix is current. To minimize convergence failures caused by an inaccurate $\mathbf{P}$, it must be updated if the coefficient $h\beta_0$ has changed significantly since the last evaluation of $\mathbf{P}$.

2.8 Interpolation at Output Stations

It frequently happens that the user requires the solution at values $\xi_{\mathrm{out},1}$, $\xi_{\mathrm{out},2}$,... of the independent variable other than the internally generated $\{\xi_n\}$. It is therefore important that in implementing the solution method provision be made for the efficient computation of the solution at the required output stations. Moreover, the procedure used for these computations should not adversely affect the efficiency of the integration beyond the output station. Such a situation arises, for example, if the method has to adjust the step size to "hit" the output station exactly. Because the Nordsieck history array is used to store past history information, the solution can be generated at the output stations quite easily, as described next.

For each ξ_{out} the integration is continued until the first mesh point n for which $\xi_n \geq \xi_{out}$, and then the solution at ξ_{out} is obtained by interpolation. Now the solution and its scaled derivatives up to order q'_{n+1} are available at ξ_n. Here q'_{n+1} is the order to be attempted on the next step, that is, $[\xi_n, \xi_{n+1}]$. Hence the solution at ξ_{out}, $\underline{Y}(\xi_{out})$, is computed by using a (q'_{n+1})th-order Taylor series expansion about ξ_n and is given by

$$\underline{Y}(\xi_{out}) = \underline{Y}_n + (\xi_{out} - \xi_n)\dot{\underline{Y}}_n + \frac{(\xi_{out} - \xi_n)^2}{2!}\ddot{\underline{Y}}_n$$

$$+ \ . \ . \ . + \frac{(\xi_{out} - \xi_n)^{q'_{n+1}}}{(q'_{n+1})!}\underline{Y}_n^{(q'_{n+1})} = \sum_{k=0}^{q'_{n+1}} \frac{(\xi_{out} - \xi_n)^k}{k!}\underline{Y}_n^{(k)}. \tag{2.116}$$

If we define the quantity r by

$$r = \frac{\xi_{out} - \xi_n}{h'_{n+1}}, \tag{2.117}$$

where h'_{n+1} is the step size to be attempted on the next step, equation (2.116) can be rewritten as

$$\underline{Y}(\xi_{out}) = \sum_{k=0}^{q'_{n+1}} r^k \frac{(h'_{n+1})^k}{k!}\underline{Y}_n^{(k)}. \tag{2.118}$$

Now

$$\frac{(h'_{n+1})^k}{k!}\underline{Y}_n^{(k)}$$

is the kth column $\underline{z}_n(k)$ of $\mathbf{z}_n$, and so equation (2.118) can be expressed compactly as

$$\underline{Y}(\xi_{\text{out}}) = \sum_{k=0}^{q'_{n+1}} r^k \underline{z}_n(k). \tag{2.119}$$

Because the solution is accurate to order q'_{n+1} at ξ_n and a (q'_{n+1})th-order Taylor series expansion is used to compute $\underline{Y}(\xi_{\text{out}})$, the latter is also accurate to order q'_{n+1}.

The solution at ξ_{out}, equation (2.119), can be evaluated by additions and multiplications alone by using Horner's rule (ref. 13):

```
$Y_i(\xi_{\text{out}}) = z_{i,n}(q'_{n+1}),$        $i = 1,...,N.$
$r = \frac{\xi_{\text{out}} - \xi_n}{h'_{n+1}}$
For $k = 1,...,q'_{n+1}$, do:                                    (2.120)
    $k' = q'_{n+1} - k$
    $Y_i(\xi_{\text{out}}) \leftarrow z_{i,n}(k') + rY_i(\xi_{\text{out}}),$        $i = 1,...,N.$
```

The Taylor series expansion method can be used to compute the solution derivative of any order (up to q'_{n+1}) at ξ_{out}. For example, the μth-derivative at ξ_{out}, $\underline{Y}^{(\mu)}(\xi_{\text{out}})$, is given by

$$\underline{Y}^{(\mu)}(\xi_{\text{out}}) = \underline{Y}_n^{(\mu)} + (\xi_{\text{out}} - \xi_n)\underline{Y}_n^{(\mu+1)} + \ . \ . \ . + \frac{(\xi_{\text{out}} - \xi_n)^{q'_{n+1}-\mu}}{(q'_{n+1} - \mu)!}\underline{Y}_n^{(q'_{n+1})}$$

$$= \sum_{k=\mu}^{q'_{n+1}} \frac{r^{k-\mu}}{(k-\mu)!}\left(h'_{n+1}\right)^{k-\mu}\underline{Y}_n^{(k)}, \tag{2.121}$$

upon using equation (2.117). Substituting $\underline{Y}_n^{(k)} = k!\underline{z}_n(k)/(h'_{n+1})^k$ into equation (2.121) produces

$$\underline{Y}^{(\mu)}(\xi_{\text{out}}) = \frac{1}{\left(h'_{n+1}\right)^{\mu}} \sum_{k=\mu}^{q'_{n+1}} r^{k-\mu}\frac{k!}{(k-\mu)!}\underline{z}_n(k). \tag{2.122}$$

2.9 Starting Procedure

At the outset of the integration, information is available at only the initial point ξ_0. Hence multistep methods cannot be used on the first step. The difficulty at the initial point is resolved easily by starting the integration with a single-step, first-order method. The Nordsieck history matrix $\mathbf{z}_0$ at ξ_0 is constructed from the initial conditions $\underline{y}_0$ and the ODE's as follows:

$$\underline{z}_0(0) \equiv \underline{Y}_0 = \underline{y}_0 \tag{2.123}$$

and

$$\underline{z}_0(1) \equiv h_0 \dot{\underline{Y}}_0 = h_0 \underline{f}(\underline{y}_0, \xi_0), \tag{2.124}$$

where h_0 is the step size to be attempted on the first step.

As the integration proceeds, the numerical solutions generated at the points ξ_1, ξ_2,... provide the necessary values for using multistep methods. Hence, as the numerical solution evolves, the method order and step size can be adjusted to their optimal values by using the procedures described in section 2.7.

Chapter 3
Description of Code

3.1 Integration and Corrector Iteration Methods

The packaged code LSODE has been designed for the numerical solution of a system of first-order ordinary differential equations (ODE's) given the initial values. It includes a variable-step, variable-order Adams-Moulton (AM) method (suitable for nonstiff problems) of orders 1 to 12 and a variable-step, variable-order backward differentiation formula (BDF) method (suitable for stiff problems) of orders 1 to 5. However, the code contains an option whereby for either method a smaller maximum method order than the default value can be specified.

Irrespective of the solution method the code starts the integration with a first-order method and, as the integration proceeds, automatically adjusts the method order (and the step size) for optimal efficiency while satisfying prescribed accuracy requirements. Both integration methods are step-by-step methods. That is, starting with the known initial condition $\underline{y}(\xi_0)$ at ξ_0, where $\underline{y}$ is the vector of dependent variables, ξ is the independent variable, and ξ_0 is its initial value, the methods generate numerical approximations $\underline{Y}_n$ to the exact solution $\underline{y}(\xi_n)$ at the discrete points ξ_n $(n = 1,2,...)$ until the end of the integration interval is reached. At each step $[\xi_{n-1},\xi_n]$ both methods employ a predictor-corrector scheme, wherein an initial guess for the solution is first obtained and then the guess is improved upon by iteration. That is, starting with an initial guess, denoted by $\underline{Y}_n^{[0]}$, successively improved estimates $\underline{Y}_n^{[m]}$ $(m = 1,...,M)$ are generated until the iteration converges, that is, further iteration produces little or no change in the solution. Here $\underline{Y}_n^{[m]}$ is the approximation computed on the mth iteration, and M is the number of iterations required for convergence.

A standard explicit predictor formula—a Taylor series expansion method devised by Nordsieck (ref. 33)—is used to generate the initial estimate for the solution. A range of iteration techniques for correcting this estimate is included in LSODE. Both the basic integration method and the corrector iteration procedure are identified by means of the method flag MF. By definition, MF has the two decimal digits METH and MITER, and

TABLE 3.1.—SUMMARY OF INTEGRATION METHODS INCLUDED IN LSODE AND CORRESPONDING VALUES OF METH, THE FIRST DECIMAL DIGIT OF MF

METH	Integration method
1	Variable-step, variable-order, implicit Adams method of orders 1 to 12
2	Variable-step, variable-order, implicit backward differentiation formula method of orders 1 to 5

TABLE 3.2.—CORRECTOR ITERATION TECHNIQUES AVAILABLE IN LSODE AND CORRESPONDING VALUES OF MITER, THE SECOND DECIMAL DIGIT OF MF

MITER	Corrector iteration technique
0	Functional iteration
1	Modified Newton iteration with user-supplied analytical Jacobian
2	Modified Newton iteration with internally generated numerical Jacobian
3	Modified Jacobi-Newton iteration with internally generated numerical Jacobian[a]
[b]4	Modified Newton iteration with user-supplied banded Jacobian
[b]5	Modified Newton iteration with internally generated banded Jacobian

[a]Modified Jacobi-Newton iteration with user-supplied analytical Jacobian can be performed by specifying MITER = 4 and ML = MU = 0[b] (i.e., a banded Jacobian with bandwidth of 1).

[b]The user must specify the lower (ML) and upper (MU) half-bandwidths of the Jacobian matrix.

$$\mathrm{MF} = 10 \times \mathrm{METH} + \mathrm{MITER}, \tag{3.1}$$

where the integers METH and MITER indicate, respectively, the integration method and the corrector iteration technique to be used on the problem. Table 3.1 summarizes the integration methods included in LSODE and the appropriate values for METH. The legal values for MITER and their meanings are given in table 3.2. The iteration procedures corresponding to MITER = 1 to 5 are described as modified Newton iteration techniques because the Jacobian matrix is not updated at every iteration.

3.2 Code Structure

The double-precision version of the LSODE package consists of the main core integration routine, LSODE, the 20 subprograms CFODE, DAXPY, DDOT, DGBFA, DGBSL, DGEFA, DGESL, DSCAL, D1MACH, EWSET, IDAMAX, INTDY, PREPJ, SOLSY, SRCOM, STODE, VNORM, XERRWV, XSETF, and

XSETUN, and a BLOCK DATA module for loading some variables. The single-precision version contains the main routine, LSODE, and the 20 subprograms CFODE, EWSET, INTDY, ISAMAX, PREPJ, R1MACH, SAXPY, SDOT, SGBFA, SGBSL, SGEFA, SGESL, SOLSY, SRCOM, SSCAL, STODE, VNORM, XERRWV, XSETF, and XSETUN. The subprograms DDOT, D1MACH, IDAMAX, ISAMAX, R1MACH, SDOT, and VNORM are function routines—all the others are subroutines. The subroutine XERRWV is machine dependent. In addition to these routines the following intrinsic and external routines are used: DABS, DFLOAT, DMAX1, DMIN1, DSIGN, and DSQRT by the double-precision version; ABS, AMAX1, AMIN1, FLOAT, SIGN, and SQRT by the single-precision version; and MAX0, MIN0, MOD, and WRITE by both versions.

Table 3.3 lists the subprograms in the order that they appear in the code and briefly describes each subprogram. Among these, the routines DAXPY, DDOT, DGBFA, DGBSL, DGEFA, DGESL, DSCAL, IDAMAX, ISAMAX, SAXPY, SDOT, SGBFA, SGBSL, SGEFA, SGESL, and SSCAL were taken from the LINPACK collection (ref. 34). The subroutines XERRWV, XSETF, and XSETUN, as used in LSODE, constitute a simplified version of the SLATEC error-handling package (ref. 35).

The structure of the LSODE package is illustrated in figure 3.1, wherein a line connecting two routines indicates that the lower routine is called by the upper one. For subprograms that have different names in the different versions of the code, both names are given, with the double-precision version name listed first. Also, the names in brackets are dummy procedure names, which are used internally and passed in call sequences. The routine F is a user-supplied subroutine that computes the derivatives $dy_i/d\xi$ $(i = 1,\ldots,N)$, where y_i is the ith component of $\underline{y}$ and N is the number of ODE's. Finally, the user-supplied subroutine JAC computes the analytical Jacobian matrix $\mathbf{J}$ $(= \partial \underline{f}/\partial \underline{y})$, where $\underline{f} = d\underline{y}/d\xi$.

The code has been arranged as much as possible in a "modular" fashion, with different subprograms performing different tasks. Hence the number of subprograms is fairly large. However, this feature aids in both understanding and, if necessary, modifying the code. To enhance the user's understanding of the code, it contains many comment statements, which are grouped together in blocks and describe both the task to be performed next and the procedure to be used. In addition, each subprogram includes detailed explanatory notes, which describe the function of the subprogram, the means of communication (i.e., call sequence and/or common blocks), and the input and output variables.

Each subprogram contains data type declarations for all variables in the routine. Such declarations are useful for debugging and provide a list of all variables that occur in a routine. This list is useful in overlay situations. For each data type the variables are usually listed in the following order: variables that are passed in the call sequence, variables appearing in common blocks, and local variables, in either alphabetical order or the order in which they appear in the call sequence and the common blocks.

TABLE 3.3.—DESCRIPTION OF SUBPROGRAMS USED IN LSODE

Subprogram		Description
Double-precision version	Single-precision version	
LSODE	LSODE	Main core integration routine. Checks legality of input, sets work array pointers, initializes work arrays, computes initial integration step size, manages solutions of ODE's, and returns to calling routine with solution and errors.
INTDY	INTDY	Computes interpolated values of the specified derivative of the dependent variables.
STODE	STODE	Advances the solution of the ODE's by one integration step. Also, computes step size and method order to be attempted on the next step.
CFODE	CFODE	Sets method coefficients for the solution and test constants for local error test and step size and method order selection.
PREPJ	PREPJ	Computes the iteration matrix and either manages the subprogram call for its LU-decomposition or computes its inverse.
SOLSY	SOLSY	Manages solution of linear system arising from chord iteration.
EWSET	EWSET	Sets the error weight vector.
VNORM	VNORM	Computes weighted root-mean-square norm of a vector.
SRCOM	SRCOM	Saves and restores contents of common blocks LS0001 and EH0001.
D1MACH	R1MACH	Computes unit roundoff of the computer.
XERRWV	XERRWV	Handles error messages.
XSETF	XSETF	Resets print control flag.
XSETUN	XSETUN	Resets logical unit number for error messages.
DGEFA	SGEFA	Performs LU-decomposition of a full matrix by Gaussian elimination.
DGESL	SGESL	Solves a linear system of equations using a previously LU-decomposed full matrix.
DGBFA	SGBFA	Performs LU-decomposition of a banded matrix by Gaussian elimination.
DGBSL	SGBSL	Solves a linear system of equations using a previously LU-decomposed banded matrix.
DAXPY	SAXPY	Forms the sum of one vector and another times a constant
DSCAL	SSCAL	Scales a vector by a constant.
DDOT	SDOT	Computes dot product of two vectors.
IDAMAX	ISAMAX	Identifies vector component of maximum absolute value.

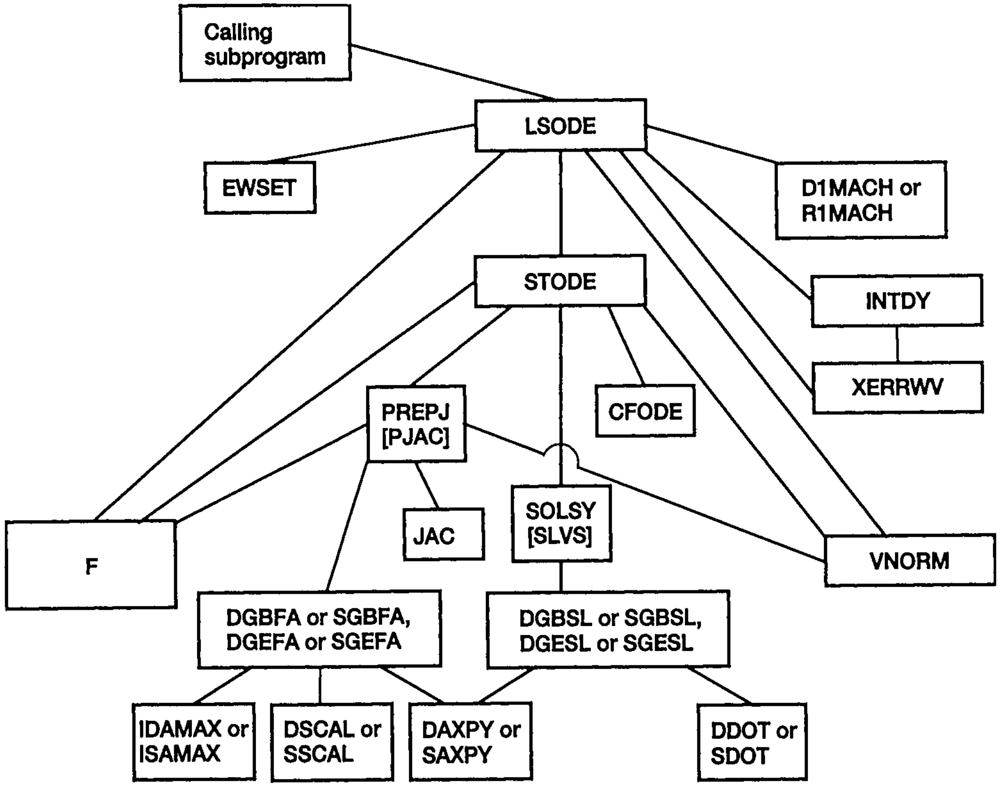


Figure 3.1.—Structure of LSODE package.

3.3 Internal Communication

Communication between different subprograms is accomplished by means of both call sequences and the two common blocks EH0001 and LS0001. The reason for using common blocks is to avoid lengthy call sequences, which can significantly deteriorate the efficiency of the program. However, common blocks are not used for variables whose dimensions are not known at compilation time. Instead, to both eliminate user adjustments to the code and minimize total storage requirements, dynamic dimensioning is used for such variables.

The common blocks, if any, used by each subprogram are given in tables 3.4 and 3.5 for the double- and single-precision versions, respectively. These tables also list all routines called and referenced (e.g., an external function) by each subprogram. Also, to facilitate use of LSODE in overlay situations, all routines that call and reference each subprogram are listed. Finally, for each subprogram the two tables give dummy procedure names (which are passed in call sequences and therefore have to be declared external in each calling and called subprogram) in brackets.

The variables included in the two common blocks and their dimensions, if different from unity, are listed in table 3.6. The common blocks contain variables that are (1) local to any routine but whose values must be preserved between calls to that routine and (2) communicated between routines. The structure of the block LS0001 is as follows: All real variables are listed first, then all integer variables. Within each group the variables are arranged in the following order: (1) those local to subroutine LSODE, (2) those local to subroutine STODE, and (3) those used for communication between routines. It must be pointed out that not all variables listed for a given common block are needed by each routine that uses it. For this reason some subprograms may use dummy names, which are not listed in table 3.6.

To further assist in user understanding and modification of the code, we have included in table 3.6 the names of all subprograms that use each common block. For the same reason we provide in tables 3.7 and 3.8 complete descriptions of the variables in EH0001 and LS0001, respectively. Also given for each variable are the default or current value, if any, and the subprogram (or subprograms) where it is set or computed. The length LENWM of the array WM in table 3.8 depends on the iteration technique and is given in table 3.9 for each legal value of MITER.

3.4 Special Features

The remainder of this chapter deals with the special features of the code and its built-in options. We also describe the procedure used to advance the solution by one step, the corrective actions taken in case of any difficulty, and step size and method order selection. In addition, we provide detailed flowcharts to explain the computational procedures. We conclude this chapter with a brief discussion of the error messages included in the code.

TABLE 3.4.—ROUTINES WITH COMMON BLOCKS, SUBPROGRAMS, AND CALLING SUBPROGRAMS IN DOUBLE-PRECISION VERSION OF LSODE

Subprogram [Dummy procedure name]	Common blocks used	Subprograms called and referenced	Calling subprograms
LSODE	LS0001	D1MACH EWSET F INTDY JAC PREPJ SOLSY STODE VNORM XERRWV	
CFODE			STODE
DAXPY			DGBFA DGBSL DGEFA DGESL
DDOT			DGBSL DGESL
DGBFA		DAXPY DSCAL IDAMAX	PREPJ
DGBSL		DAXPY DDOT	SOLSY
DGEFA		DAXPY DSCAL IDAMAX	PREPJ
DGESL		DAXPY DDOT	SOLSY
DSCAL			DGBFA DGEFA
D1MACH			LSODE
EWSET			LSODE
IDAMAX			DGBFA DGEFA
INTDY	LS0001	XERRWV	LSODE
PREPJ [PJAC]	LS0001	DGBFA DGEFA F JAC VNORM	STODE
SOLSY [SLVS]	LS0001	DGBSL DGESL	STODE
SRCOM	EH0001 LS0001		
STODE	LS0001	CFODE F JAC PREPJ SOLSY VNORM	LSODE
VNORM			LSODE PREPJ STODE
XERRWV	EH0001		LSODE INTDY
XSETF	EH0001		
XSETUN	EH0001		
BLOCK DATA	EH0001 LS0001		

TABLE 3.5.—ROUTINES WITH COMMON BLOCKS, SUBPROGRAMS, AND CALLING SUBPROGRAMS IN SINGLE-PRECISION VERSION OF LSODE

Subprogram [Dummy procedure name]	Common blocks used	Subprograms called and referenced	Calling subprograms
LSODE	LS0001	EWSET F INTDY JAC PREPJ R1MACH SOLSY STODE VNORM XERRWV	
CFODE			STODE
EWSET			LSODE
INTDY	LS0001	XERRWV	LSODE
ISAMAX			SGBFA SGEFA
PREPJ [PJAC]	LS0001	F JAC SGBFA SGEFA VNORM	STODE
R1MACH			LSODE
SAXPY			SGBFA SGBSL SGEFA SGESL
SDOT			SGBSL SGESL
SGBFA		ISAMAX SAXPY SSCAL	PREPJ
SGBSL		SAXPY SDOT	SOLSY
SGEFA		ISAMAX SAXPY SSCAL	PREPJ
SGESL		SAXPY SDOT	SOLSY
SOLSY [SLVS]	LS0001	SGBSL SGESL	STODE
SRCOM	EH0001 LS0001		
SSCAL			SGBFA SGEFA
STODE	LS0001	CFODE F JAC PREPJ SOLSY VNORM	LSODE
VNORM			LSODE PREPJ STODE
XERRWV	EH0001		LSODE INTDY
XSETF	EH0001		
XSETUN	EH0001		

TABLE 3.6.—COMMON BLOCKS WITH VARIABLES AND SUBPROGRAMS WHERE USED

Common block	Variables (dimension)	Subprograms where used
EH0001	MESFLG LUNIT	SRCOM XERRWV XSETF XSETUN BLOCK DATA[a]
LS0001	CONIT CRATE EL(13) ELCO(13, 12) HOLD RMAX TESCO(3, 12) CCMAX EL0 H HMIN HMXI HU RC TN UROUND ILLIN INIT LYH LEWT LACOR LSAVF LWM LIWM MXSTEP MXHNIL NHNIL NTREP NSLAST NYH IALTH IPUP LMAX MEO NQNYH NSLP ICF IERPJ IERSL JCUR JSTART KFLAG L METH MITER MAXORD MAXCOR MSBP MXNCF N NQ NST NFE NJE NQU	LSODE INTDY PREPJ SOLSY SRCOM STODE BLOCK DATA[a]

[a]Double-precision version only.

TABLE 3.7.—DESCRIPTION OF VARIABLES IN COMMON BLOCK EH0001, THEIR CURRENT VALUES, AND SUBPROGRAMS WHERE THEY ARE SET

Variable	Description	Current value	Subprogram where variable is set
MESFLG	Integer flag, which controls printing of error messages from code and has following values and meanings: 0 No error message is printed. 1 All error messages are printed.	1	BLOCK DATA in double-precision version and XERRWV in single-precision version
LUNIT	Logical unit number for messages from code	6	BLOCK DATA in double-precision version and XERRWV in single-precision version

TABLE 3.8.—DESCRIPTION OF VARIABLES IN COMMON BLOCK LS0001, THEIR CURRENT VALUES, IF ANY, AND SUBPROGRAMS WHERE THEY ARE SET OR COMPUTED[a]

Variable	Description	Current value, if any	Subprograms where variable is set or computed
CONIT	Empirical factor, 0.5/(NQ + 1) used in convergence test (see eq. (2.99))	------------------------	STODE
CRATE	Estimated convergence rate of iteration	------------------------	STODE
EL	Method coefficients in normal form $\{\ell_i\}$ (see eq. (2.68)), for current method order	------------------------	STODE
ELCO	Method coefficients in normal form for current method of orders 1 to MAXORD	------------------------	CFODE
HOLD	Step size used on last successful step or attempted on last unsuccessful step	------------------------	STODE
RMAX	Maximum factor by which step size will be increased when step size change is next considered	Normally 10; 10^4 for very first step size increase for problem if no difficulty encountered; 2 after a failed convergence or local error test	STODE
TESCO	Test coefficients for current method of orders 1 to MAXORD; used for testing convergence and local accuracy and selecting new step size and method order	------------------------	CFODE
CCMAX	Maximum relative change allowed in H×EL0 before Jacobian matrix is updated	0.3	LSODE
EL0	Method coefficient ℓ_0 (see eq. (2.68)) for current method and current order	------------------------	STODE
H	Step size either being used on this step or to be attempted on next step	------------------------	LSODE STODE
HMIN[b]	Minimum absolute value of step size to be used on any step	0.0	
HMXI[b]	Inverse of maximum absolute value of step size to be used on any step	0.0	LSODE
HU	Step size used on last successful step	------------------------	STODE

[a]Note that some variables appear in the table before they are defined.
[b]Default value for this variable can be changed by the user, as described in table 4.6.

TABLE 3.8.—Continued.

Variable	Description	Current value, if any	Subprograms where variable is set or computed
RC	Relative change in H×EL0 since last update of Jacobian matrix	----------------------	STODE
TN	Value of independent variable to which integrator either has successfully advanced solution or will do so after next step	----------------------	STODE
UROUND	Unit roundoff of computer	----------------------	D1MACH in double-precision version and R1MACH in single-precision version
ILLIN	Number of consecutive times LSODE has been called with illegal input for current problem	----------------------	Initialized in BLOCK DATA (double-precision version) and LSODE (single-precision version). Updated in LSODE in both versions.
INIT	Integer flag (= 0 or 1) that denotes if initialization of LSODE has been performed (INIT = 1) or not (INIT = 0)	----------------------	LSODE
LYH	Base address for Nordsieck history array YH of length NYH×(MAXORD + 1)	21	LSODE
LEWT	Base address for error weight vector EWT of length N	LWM + LENWM[c]	LSODE
LACOR	Base address for array ACOR (of length N) containing local errors on last successful step	LEWT + 2N	LSODE
LSAVF	Base address for an array SAVF (of length N), used for temporary storage	LEWT + N	LSODE
LWM	Base address for array WM (of length LENWM[c]), required for linear algebra associated with Jacobian and iteration matrices	LYH + NYH×(MAXORD + 1)	LSODE

[c]The length LENWM of the array WM depends on the iteration technique and is given in table 3.9.

TABLE 3.8.—Continued.

Variable	Description	Current value, if any	Subprograms where variable is set or computed
LIWM	Base address for integer work array IWM	1	LSODE
MXSTEP[b]	Maximum number of steps allowed on any one call to LSODE	500	LSODE
MXHNIL[b]	Maximum number of times that warning message that step size is so small that TN + H = TN for next step is printed	10	LSODE
NHNIL	Number of times that this difficulty with small step size has been encountered so far for problem	----------------------	LSODE
NTREP	Number of consecutive times an initialization or "first" call (see table 4.3) has been made to LSODE with same initial and final values for integration interval	----------------------	Initialized in BLOCK DATA (double-precision version) and LSODE (single-precision version). Updated in LSODE in both versions.
NSLAST	Number of steps used for problem prior to current call to LSODE; used to check that the limit of MXSTEP steps is not exceeded	----------------------	LSODE
NYH	Maximum number of ODE's to be solved for current problem (This number is equal to the number of ODE's specified on first call to LSODE.)	----------------------	LSODE
IALTH	Integer counter, related to step size and method order changes, with following values and meanings: 0 Select optimal step size and method order. 1 If NQU < MAXORD, save vector $\underline{e}$ (see eqs. (2.76) and (2.111)) so that an order increase can be considered on the next step. >1 Neither of these two operations is to be performed.	----------------------	STODE

[b]Default value for this variable can be changed by the user, as described in table 4.6.

TABLE 3.8.—Continued.

Variable	Description	Current value, if any	Subprograms where variable is set or computed
IPUP	Integer flag, related to Jacobian matrix update, with following values and meanings: 0 Jacobian matrix is either not needed or does not have to be updated. >0 Jacobian matrix must be updated before corrector iteration.	------------------------	STODE
LMAX	Maximum number of columns of Nordsieck history array	MAXORD + 1	STODE
MEO	Integration method specified on previous call to LSODE	------------------------	STODE
NQNYH	Number of elements of Nordsieck history array that are changed by predictor	NQ×NYH	STODE
NSLP	Step number when Jacobian matrix was last updated	------------------------	STODE
ICF	An integer flag, related to iteration convergence, with following values and meanings: 0 Solution converged. 1 Convergence test failed and Jacobian matrix is not current. 2 Convergence test failed and Jacobian matrix is either current or not needed.	------------------------	STODE
IERPJ	Integer flag, related to singularity of iteration matrix, with following values and meanings: 0 Iteration matrix was successfully LU-decomposed (MITER = 1, 2, 4, or 5) or inverted (MITER = 3) (see table 3.2) 1 Iteration matrix was found to be singular.	------------------------	PREPJ
IERSL	Integer flag, related to singularity of interation matrix modified to account for new (H×EL0) for MITER = 3 (see table 3.2). IERSL has following values and meanings: 0 Modified iteration matrix was successfully inverted and corrections computed. 1 New matrix was found to be singular.	------------------------	SOLSY

TABLE 3.8.—Continued.

Variable	Description	Current value, if any	Subprograms where variable is set or computed
JCUR	Integer flag, related to state of Jacobian matrix, with following values and meanings: 0 Jacobian matrix is not current and may need to be updated later. 1 Matrix is current.	--------------------	PREPJ STODE
JSTART	Integer flag, used to communicate state of calculation to STODE, with following values and meanings: 0 This is the first step for the problem. 1 Continue normal calculation of problem. (This is the value returned by STODE to facilitate continuation.) −1 Take the next step with new values for H, MAXORD, N, METH (see table 3.1), MITER (see table 3.2), and/or matrix parameters.	--------------------	LSODE STODE
KFLAG	A completion code from STODE with following values and meanings: 0 Step was successful. −1 Requested local accuracy in solution could not be achieved. −2 Repeated convergence test failures occurred.	--------------------	STODE
L	Number of columns of Nordsieck array	NQ + 1	STODE
METH	Integration method to be used on next step	--------------------	LSODE
MITER	Iteration technique to be used on next step	--------------------	LSODE
MAXORD[b]	Maximum method order to be used for problem	12 for Adams-Moulton method and 5 for backward differentiation formula method	LSODE
MAXCOR	Maximum number of corrector iterations to be attempted on any one step	3	LSODE
MSBP	Maximum number of steps for which same Jacobian matrix is used	20	LSODE

[b]Default value for this variable can be changed by the user, as described in table 4.6.

TABLE 3.8.—Concluded.

Variable	Description	Current value, if any	Subprograms where variable is set or computed
MXNCF	Maximum number of corrector convergence failures allowed on any one step	10	LSODE
N	Number of ODE's to be solved on next step	-------	-------
NQ	Method order either being tried on this step or to be attempted on next step	-------	STODE
NST	Total number of integration steps used so far for problem	-------	LSODE STODE
NFE	Total number of derivative evaluations required so far for problem	-------	LSODE STODE
NJE	Total number of Jacobian matrix evaluations (and iteration matrix LU-decompositions or inversions) required so far for problem	-------	LSODE PREPJ
NQU	Method order used on last successful step.	-------	STODE

TABLE 3.9.—LENGTH LENWM OF ARRAY WM IN TABLE 3.8 FOR ITERATION TECHNIQUES INCLUDED IN CODE

MITER[a]	LENWM[b]
0	0
1,2	$N^2 + 1$
3	$N + 2$
4,5	$(2ML + MU + 1)N + 2$

[a]See table 3.2 for description of MITER.
[b]N is the number of ODE's and ML and MU are defined in table 3.2.

3. Description of Code

The main routine, LSODE, controls the integration and serves as an interface between the calling subprogram and the rest of the package. A flowchart of this subroutine is given in figure 3.2. In this figure ITASK and ISTATE are user-specified integers that specify, respectively, the task to be performed and the state of the calculation, that is, if the call to LSODE is the first one for the problem or a continuation; if the latter, ISTATE further indicates if the continuation is a normal one or if the user has changed one or more parameters since the last call to LSODE (see chapter 4 for details). On return from LSODE the value of ISTATE indicates if the integration was performed successfully, and if not, the reason for failure. The integer JSTART is an internally defined variable used for communicating the state of the calculation with the routine STODE. The variables T (= ξ), H, and $\underline{Y}$ are, respectively, the independent variable, the step size to be attempted on the next step, and the numerical solution vector. TOUT is the ξ value at which the solution is next required. Finally, TCRIT is the ξ value that the integrator must not overshoot. This option is useful if a singularity exists at or beyond TCRIT and is discussed further in chapter 4.

The subroutine STODE advances the numerical solution to the ODE's by a single integration step $[\xi_{n-1}, \xi_n]$. It also computes the method order and step size to be attempted on the next step. The efficiency of the integration procedure is increased by saving the solution history, which is required by the multistep methods used in the code, in the form suggested by Nordsieck (ref. 33). The Nx$(q + 1)$ Nordsieck history matrix $\mathbf{z}_{n-1}$ at ξ_{n-1} contains the numerical solution $\underline{Y}_{n-1}$ and the q scaled derivatives $h_n^j \underline{Y}_{n-1}^{(j)}/j!$ ($j = 1,...,q$), where h_n ($= \xi_n - \xi_{n-1}$) and q are, respectively, the current step size and method order and $\underline{Y}^{(j)} = d^j\underline{Y}/d\xi^j$.

The flowchart of STODE is presented in figure 3.3. In this figure NCF is the number of corrector convergence failures on the current step, KFLAG is an internally defined integer used for communication with LSODE, NQ (= q) is the method order to be attempted on the current step, and the integer counter IALTH indicates how many more steps are to be taken with the current step size and method order. The (NQ + 1)-dimensional vector $\underline{\ell}$ contains the method coefficients and depends on both the integration method and the method order; ℓ_0 is the zeroth component of $\underline{\ell}$ (see eq. (2.68)). The matrix $\mathbf{z}_n^{[0]}$ is the predicted Nordsieck history matrix at ξ_n, and the NxN iteration matrix $\mathbf{P}$ is given by equation (2.25). The variable R is the ratio of the step size to be attempted next to its current value, RMAX is the maximum R allowed when a step size change is next considered, and HMIN and HMAX are user-supplied minimum and maximum absolute values for the step size to be tried on any step. The ratios RHDN, RHSM, and RHUP are factors by which the step size can be increased if the new method order is NQ $-$ 1, NQ (the current value), and NQ + 1, respectively. Finally, NQMAX is the maximum method order that may be attempted on any step, and the vector $\underline{e}_n$ ($= h_n\dot{\underline{Y}}_n - h_n\dot{\underline{Y}}_n^{[0]}$) is proportional to the local truncation error vector at ξ_n (see eqs. (2.87) and (2.89)).

3.4.1 Initial Step Size Calculation

An important feature of LSODE is that it will compute the step size h_0 to be attempted on the first step if the user does not provide a value for it. The calculation procedure attempts to produce an h_0 such that the numerical solution $\underline{Y}_1$ generated at the first internal mesh point ξ_1 will satisfy the local error test. Now with either solution technique the code starts the integration with a first-order method. Hence the asymptotic local truncation error $d_{i,1}$ in the ith solution component at ξ_1 will be equal to $(1/2)h_1^2\ddot{y}_i(\xi_1)$ for both the AM and BDF methods of order 1. Here h_1 is the step size successfully used on the first step, and $\ddot{y}_i(\xi_1)$ is the second derivative of the ith component of $\underline{y}$ at ξ_1. To pass the local error test, equation (2.91), the weighted local error vector, that is, $\{d_{i,1}/\text{EWT}_{i,1}\}$, must satisfy the inequality

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{1}{2}h_1^2\ddot{y}_i(\xi_1)}{\text{EWT}_{i,1}}\right)^2} \leq 1, \tag{3.2}$$

where $\text{EWT}_{i,1}$ is the ith component of the error weight vector for the first step (see eq. (2.90)):

$$\text{EWT}_{i,1} = \text{RTOL}_i\left|Y_{i,0}\right| + \text{ATOL}_i. \tag{3.3}$$

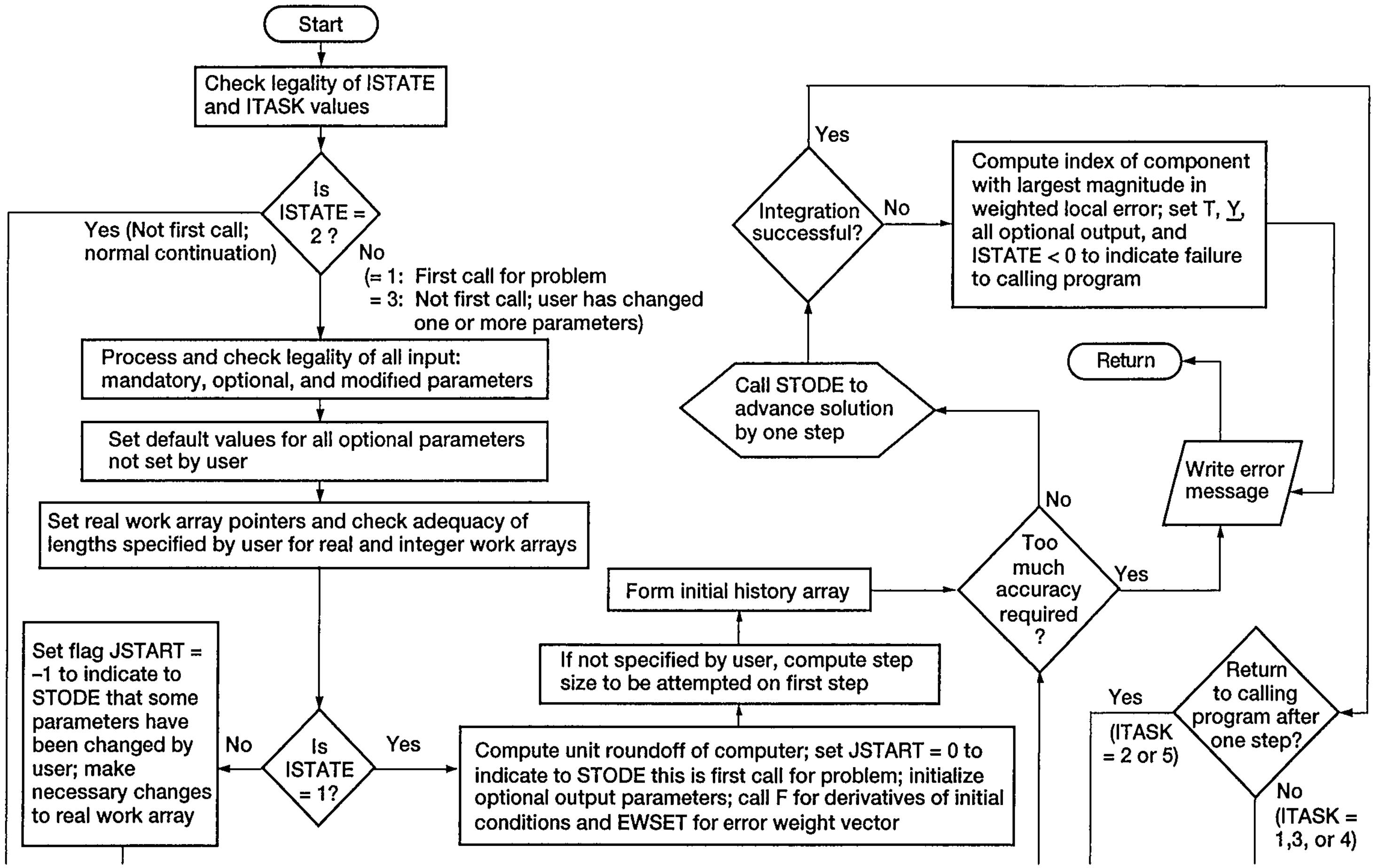
In this equation RTOL_i and ATOL_i are, respectively, the user-supplied local relative and absolute error tolerances for the ith solution component, $Y_{i,0}$ is the ith solution component at ξ_0, and the vertical bars $|\bullet|$ denote absolute value.

The test given by equation (3.2) cannot be applied at the start of the step $[\xi_0, \xi_1]$ because $\ddot{\underline{y}}(\xi_1)$ is not known. We therefore modify this test by using $\ddot{\underline{y}}(\xi_0)$ as follows: We first define a weighted principal error function at order 1, $\underline{\phi}$, with element ϕ_i given by

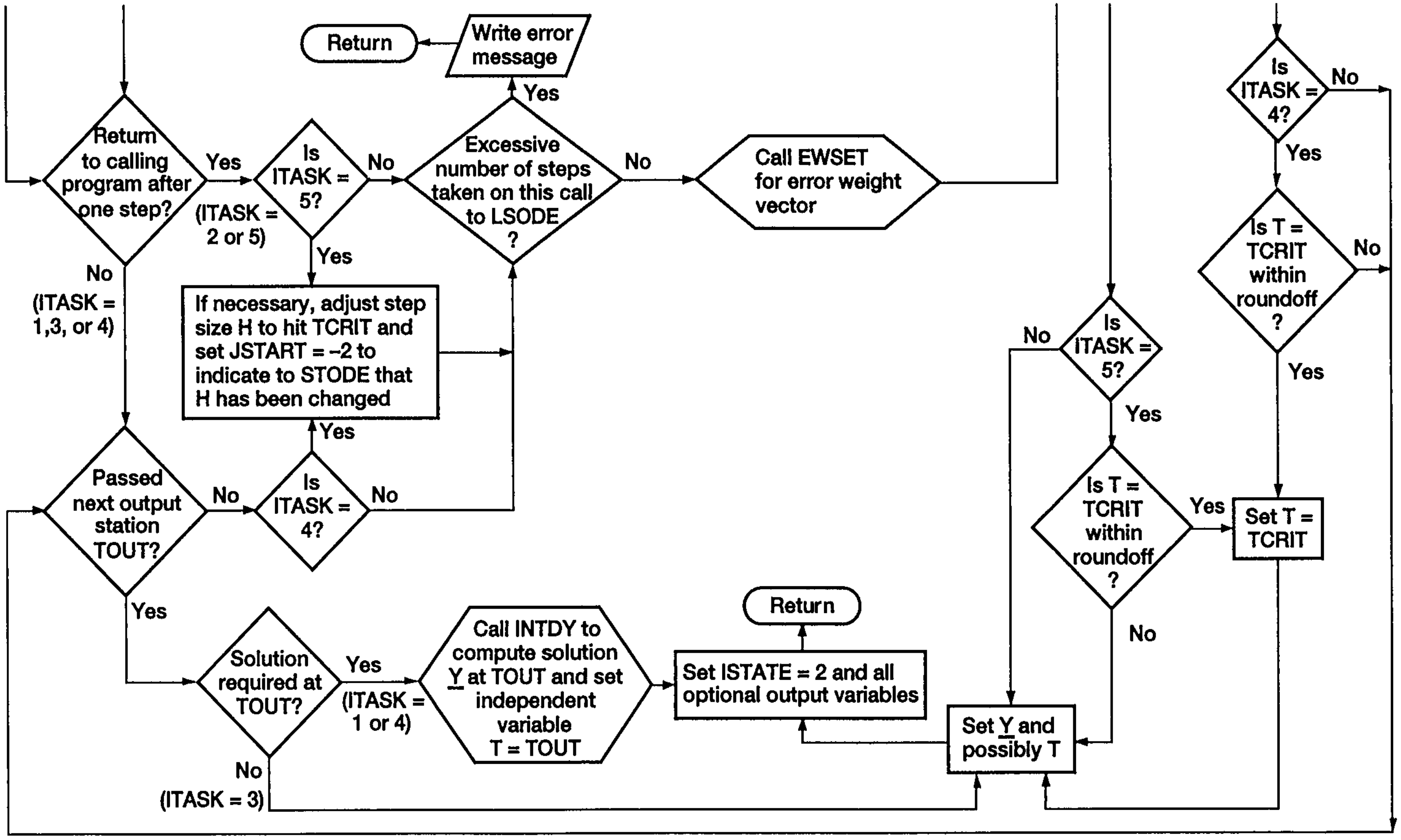
$$\phi_i = \frac{1}{2}\frac{\ddot{y}_i(\xi_0)}{W_i}, \tag{3.4}$$

where

$$W_i = \text{EWT}_{i,1}/\text{TOL}, \tag{3.5}$$

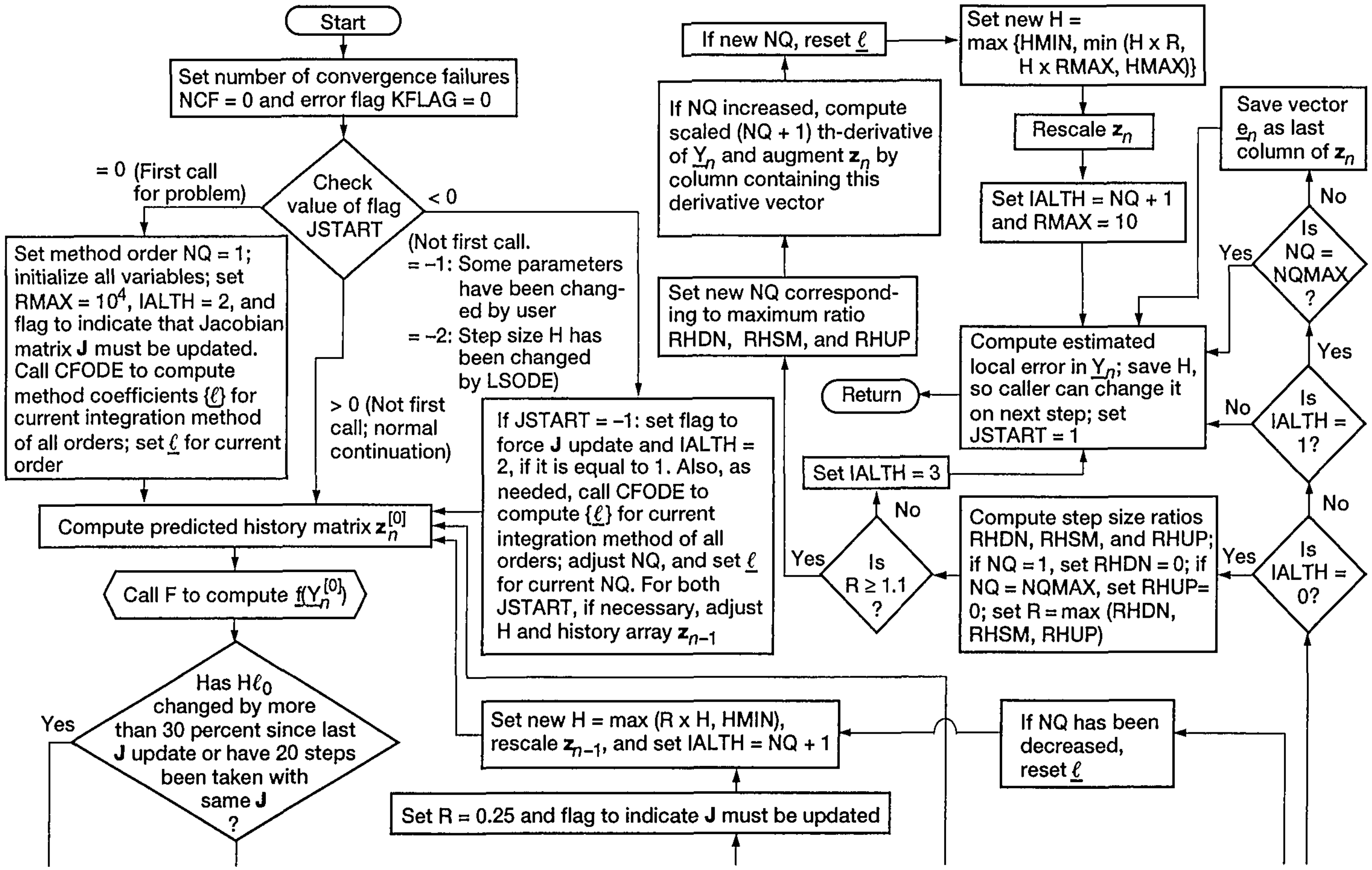


Start
Check legality of ISTATE and ITASK values
Is ISTATE = 2 ?
Yes (Not first call; normal continuation)
No (= 1: First call for problem = 3: Not first call; user has changed one or more parameters)
Process and check legality of all input: mandatory, optional, and modified parameters
Set default values for all optional parameters not set by user
Set real work array pointers and check adequacy of lengths specified by user for real and integer work arrays
Is ISTATE = 1?
No
Yes
Set flag JSTART = –1 to indicate to STODE that some parameters have been changed by user; make necessary changes to real work array
Compute unit roundoff of computer; set JSTART = 0 to indicate to STODE this is first call for problem; initialize optional output parameters; call F for derivatives of initial conditions and EWSET for error weight vector
If not specified by user, compute step size to be attempted on first step
Form initial history array
Too much accuracy required ?
No
Yes
Call STODE to advance solution by one step
Integration successful?
Yes
No
Compute index of component with largest magnitude in weighted local error; set T, Y, all optional output, and ISTATE < 0 to indicate failure to calling program
Write error message
Return
Return to calling program after one step?
Yes (ITASK = 2 or 5)
No (ITASK = 1,3, or 4)



Return
Write error message
Yes
Return to calling program after one step?
Yes
Is ITASK = 5?
No
Excessive number of steps taken on this call to LSODE ?
No
Call EWSET for error weight vector
(ITASK = 2 or 5)
Yes
No (ITASK = 1,3, or 4)
If necessary, adjust step size H to hit TCRIT and set JSTART = −2 to indicate to STODE that H has been changed
Yes
Passed next output station TOUT?
No
Is ITASK = 4?
No
Yes
Solution required at TOUT?
Yes (ITASK = 1 or 4)
Call INTDY to compute solution Y at TOUT and set independent variable T = TOUT
Return
Set ISTATE = 2 and all optional output variables
No (ITASK = 3)
Is ITASK = 4?
No
Yes
Is T = TCRIT within roundoff ?
No
Yes
No
Is ITASK = 5?
Yes
Is T = TCRIT within roundoff ?
Yes
Set T = TCRIT
No
Set Y and possibly T

Figure 3.2.—Flowchart of subroutine LSODE.


Start
Set number of convergence failures NCF = 0 and error flag KFLAG = 0
Check value of flag JSTART
= 0 (First call for problem)
< 0
(Not first call.
= –1: Some parameters have been changed by user
= –2: Step size H has been changed by LSODE)
> 0 (Not first call; normal continuation)
Set method order NQ = 1; initialize all variables; set RMAX = 10^4, IALTH = 2, and flag to indicate that Jacobian matrix **J** must be updated. Call CFODE to compute method coefficients {ℓ} for current integration method of all orders; set $\underline{\ell}$ for current order
If JSTART = –1: set flag to force **J** update and IALTH = 2, if it is equal to 1. Also, as needed, call CFODE to compute {ℓ} for current integration method of all orders; adjust NQ, and set $\underline{\ell}$ for current NQ. For both JSTART, if necessary, adjust H and history array $\mathbf{z}_{n-1}$
Compute predicted history matrix $\mathbf{z}_n^{[0]}$
Call F to compute $\underline{f}(\underline{Y}_n^{[0]})$
Has $H\ell_0$ changed by more than 30 percent since last **J** update or have 20 steps been taken with same **J** ?
Yes
Set new H = max (R x H, HMIN), rescale $\mathbf{z}_{n-1}$, and set IALTH = NQ + 1
Set R = 0.25 and flag to indicate **J** must be updated
If NQ has been decreased, reset $\underline{\ell}$
If new NQ, reset $\underline{\ell}$
If NQ increased, compute scaled (NQ + 1) th-derivative of $\underline{Y}_n$ and augment $\mathbf{z}_n$ by column containing this derivative vector
Set new NQ corresponding to maximum ratio RHDN, RHSM, and RHUP
Set new H = max {HMIN, min (H x R, H x RMAX, HMAX)}
Rescale $\mathbf{z}_n$
Set IALTH = NQ + 1 and RMAX = 10
Save vector $\underline{e}_n$ as last column of $\mathbf{z}_n$
Is NQ = NQMAX ?
No
Yes
Compute estimated local error in $\underline{Y}_n$; save H, so caller can change it on next step; set JSTART = 1
Return
Is IALTH = 1?
No
Yes
Set IALTH = 3
Is R ≥ 1.1 ?
No
Yes
Compute step size ratios RHDN, RHSM, and RHUP; if NQ = 1, set RHDN = 0; if NQ = NQMAX, set RHUP= 0; set R = max (RHDN, RHSM, RHUP)
Is IALTH = 0?
Yes
No

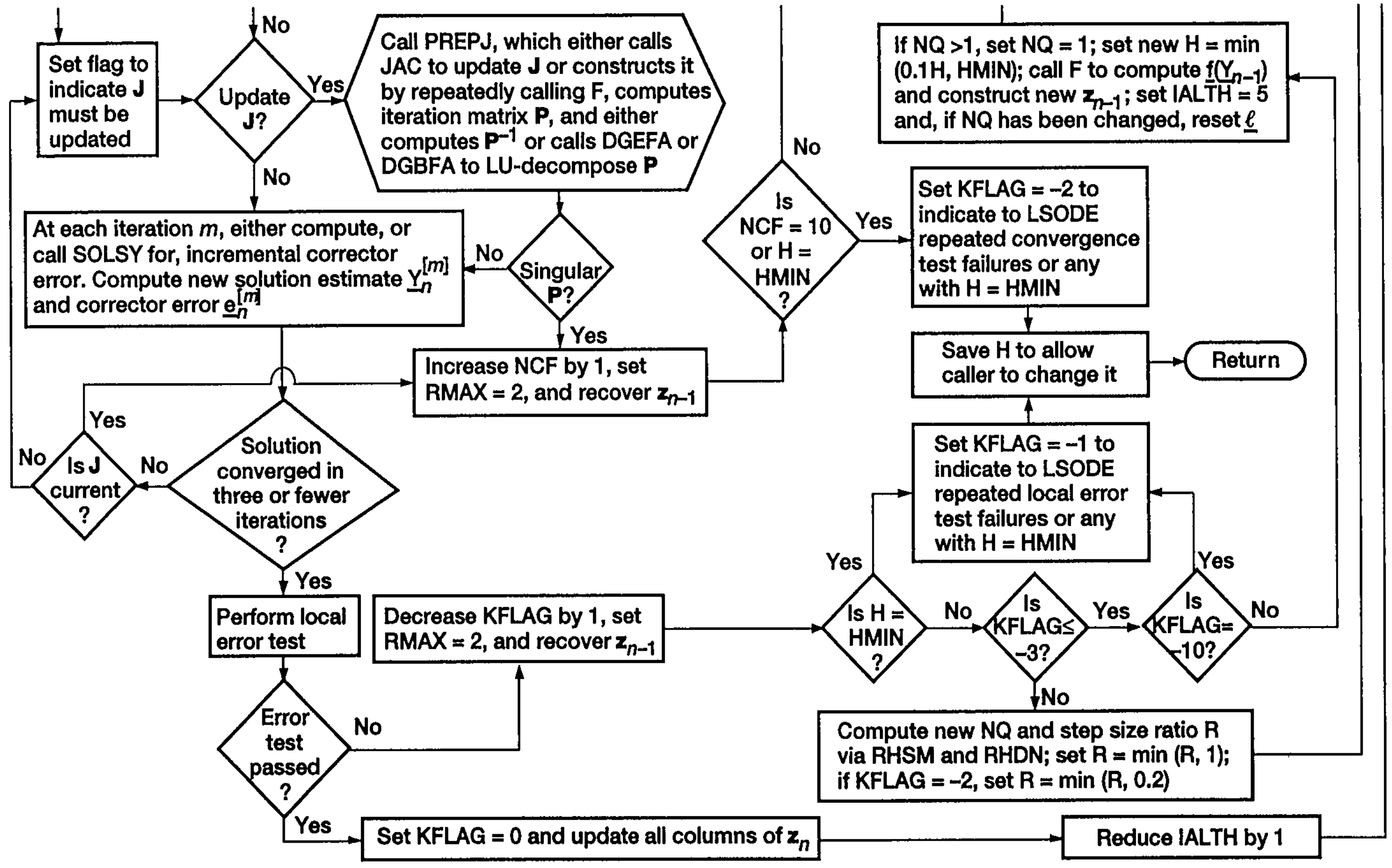

Set flag to indicate J must be updated
No
Update J?
Yes
No
Call PREPJ, which either calls JAC to update J or constructs it by repeatedly calling F, computes iteration matrix P, and either computes P^{-1} or calls DGEFA or DGBFA to LU-decompose P
Singular P?
No
Yes
At each iteration m, either compute, or call SOLSY for, incremental corrector error. Compute new solution estimate $\underline{Y}_n^{[m]}$ and corrector error $\underline{e}_n^{[m]}$
Increase NCF by 1, set RMAX = 2, and recover z_{n-1}
Is NCF = 10 or H = HMIN ?
No
Yes
Set KFLAG = −2 to indicate to LSODE repeated convergence test failures or any with H = HMIN
Save H to allow caller to change it
Return
If NQ >1, set NQ = 1; set new H = min (0.1H, HMIN); call F to compute $\underline{f}(\underline{Y}_{n-1})$ and construct new z_{n-1}; set IALTH = 5 and, if NQ has been changed, reset $\underline{\ell}$
Yes
No
Is J current ?
No
Solution converged in three or fewer iterations ?
Yes
Perform local error test
Error test passed ?
No
Yes
Decrease KFLAG by 1, set RMAX = 2, and recover z_{n-1}
Set KFLAG = −1 to indicate to LSODE repeated local error test failures or any with H = HMIN
Yes
Is H = HMIN ?
No
Is KFLAG≤ −3?
Yes
Is KFLAG= −10?
Yes
No
No
Compute new NQ and step size ratio R via RHSM and RHDN; set R = min (R, 1); if KFLAG = −2, set R = min (R, 0.2)
Set KFLAG = 0 and update all columns of z_n
Reduce IALTH by 1

Figure 3.3.—Flowchart of subroutine STODE.

and the scalar tolerance quantity TOL, which is to be determined, is such that W_i is a suitable weight for Y_i, the ith component of $\underline{Y}$. The step size and the local error are then together required to satisfy the inequality

$$h_0^2 \|\underline{\phi}\| \le \text{TOL}, \tag{3.6}$$

where $\|\bullet\|$ represents a suitable norm. We have used a different symbol for the initial step size than in equation (3.2) to indicate that this quantity is not known and must be computed. Because a first-order method will be used on this step, for a sufficiently small step size the numerical approximation $\underline{\ddot{Y}}_1$ at ξ_1 will not be significantly different from $\underline{\ddot{y}}(\xi_0)$, and use of the latter quantity is therefore reasonable. The rationale for introducing TOL will become apparent shortly.

The second derivative $\underline{\ddot{y}}(\xi_0)$ is not generally available, and so the following empirical procedure is used to estimate it. We consider the dominant eigenvalue ($= \lambda$) of the ODE system and model this component with the simple scalar ODE

$$\dot{y} \equiv \frac{dy}{d\xi} = \lambda y, \tag{3.7}$$

where $|\lambda| >> 1$. For this problem, $\phi = (1/2)\ddot{y}/W = (1/2)\lambda^2 y/W$. Now, if TOL is chosen such that y/W is of order unity, ϕ can be approximated by $(\dot{y}/W)^2$ [$= (\lambda y/W)^2$],which is known. For the scalar ODE this condition is obtained by setting TOL = RTOL and ATOL = 0 (see eqs. (3.3) and (3.5)). The quantity $\dot{y}/W$ may be regarded as the weighted principal error function for a "zeroth order" method. We use this empirical rule to replace each ϕ_i by $(\dot{y}_i/W_i)^2$ so that equation (3.6) can be written as

$$h_0^2 \left\| \left[\left(f_{i,0}/W_i \right)^2 \right]_1^N \right\| \le \text{TOL}, \tag{3.8}$$

where $f_{i,0}$ [$=f_i(\underline{Y}_0, \xi_0)$] is the first derivative of the ith component at ξ_0. Because the weighted root-mean-square (rms) norm is used in the local error test, equation (3.2), for convenience, we use the following criterion for initial step size control:

$$h_0^2 \left[\frac{1}{N} \sum_{i=1}^{N} \left(\frac{f_{i,0}}{W_i} \right)^2 \right] \le \text{TOL}. \tag{3.9}$$

Equations (3.5) and (3.9) together show that h_0 ($\propto 1/\sqrt{\text{TOL}}$) is a decreasing function of TOL. To produce a reliable estimate for h_0, we therefore select a TOL erring on the high side. A suitable value is given by

$$\mathrm{TOL} = \max_i(\mathrm{RTOL}_i). \tag{3.10}$$

This expression cannot be used if all $\mathrm{RTOL}_i = 0$. In this case an appropriate value for TOL is given by

$$\mathrm{TOL} = \max_i\left(\frac{\mathrm{ATOL}_i}{|Y_{i,0}|}\right) \quad \text{for } Y_{i,0} \neq 0. \tag{3.11}$$

In any case the value of TOL is constrained to be within reasonable bounds as follows:

$$100u \le \mathrm{TOL} \le 10^{-3}, \tag{3.12}$$

where u is the unit roundoff of the computer or the machine epsilon (ref. 13). It is the smallest positive number such that $1 + u > 1$.

Equation (3.9) cannot be used to compute h_0 if either each $f_{i,0}$ is equal to zero or the norm is very small. To produce a reasonable h_0 in such an event, we include the independent variable ξ as the zeroth component y_0 of $\underline{y}$ and modify equation (3.9) as follows:

$$h_0^2\left[\frac{1}{W_0^2} + \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f_{i,0}}{W_i}\right)^2\right] \le \mathrm{TOL}, \tag{3.13}$$

where we have used the fact that $\dot{y}_0 = 1$. To be consistent with the other W_i, which are of order $Y_{i,0}$, the weight W_0 should be of order ξ_0; however, we use

$$W_0 = \max\left(|\xi_0|, |\xi_{\mathrm{out},1}|\right) \tag{3.14}$$

to ensure that it is not equal to zero. In equation (3.14), $\xi_{\mathrm{out},1}$ is either the first (or only) value of the independent variable at which the solution is required or, as discussed in chapter 4, a value that gives both the direction of integration (i.e., increasing or decreasing ξ) and an approximate scale of the problem. If the quantity $\xi_{\mathrm{out},1} - \xi_0$ is not significantly different from zero, an error exit occurs. Equation (3.13) gives a reasonable value for h_0 $(= W_0\sqrt{\mathrm{TOL}})$ if $\underline{f}_0 = 0$.

The calculation procedure used for h_0 is therefore given by

$$h_0 = \sqrt{\frac{\text{TOL}}{W_0^{-2} + \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f_{i,0}}{W_i}\right)^2}} \tag{3.15}$$

Several restrictions apply to the step size given by equation (3.15). It is not allowed to be greater than the difference $|\xi_{\text{out},1} - \xi_0|$. Hence

$$h_0 \leftarrow \min\left(h_0, \left|\xi_{\text{out},1} - \xi_0\right|\right). \tag{3.16}$$

In addition, if the user has supplied a value for $h_{\max}$, the maximum step size to be used on any step, h_0 is restricted to

$$h_0 \leftarrow \min\left(h_0, h_{\max}\right). \tag{3.17}$$

However, no comparison of h_0 is made with $h_{\min}$, the user-supplied minimum step size to be used on any step, so that h_0 is allowed to be less than $h_{\min}$. Finally the sign of h_0 is adjusted to reflect the direction of integration.

3.4.2 Switching Methods

Another useful feature of LSODE is that different integration methods and/or different iteration techniques can be used in different subintervals of the problem. This option is useful when the problem changes character and is stiff in some regimes and nonstiff in others as, for example, in combustion chemistry. Indeed, because stiff problems are usually characterized by a nonstiff initial "transient" region, the ability to switch integration methods is a desirable feature of any ODE package. During the course of solving a problem the method flag MF may be changed both whenever and as many times as desired. As described in chapter 4 changing methods is quite straightforward.

3.4.3 Excessive Accuracy Specification Test

At each integration step $[\xi_{n-1}, \xi_n]$ LSODE checks that the user has not requested too much accuracy for the precision of the machine. This condition is said to occur if the criterion

$$d_{i,n} \overset{?}{<} uY_{i,n} \tag{3.18}$$

is true for all N solution components. In equation (3.18), $d_{i,n}$ is the estimated local truncation error in $Y_{i,n}$, the ith solution component at ξ_n. Now the numerical solution $\underline{Y}_n$ at ξ_n is judged to be sufficiently accurate if the following inequality is satisfied (see chapter 2):

$$\|\underline{d}_n\| \equiv \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{d_{i,n}}{\mathrm{EWT}_{i,n}}\right)^2} \overset{?}{\leq} 1. \tag{2.91}$$

The quantity $\mathrm{EWT}_{i,n}$ is the ith component of the error weight vector, equation (2.90), for this step. Equations (3.18) and (2.91) together imply that if the quantity TOLSF (tolerance scale factor) defined as

$$\mathrm{TOLSF} = u\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{Y_{i,n}}{\mathrm{EWT}_{i,n}}\right)^2} \tag{3.19}$$

is greater than 1, the test for excessive accuracy requirement is passed. This test is quite inexpensive, but it can be applied only after the solution at ξ_n is produced. It is, however, wasteful to generate a solution only to discover that excessive accuracy has been required, either because TOLSF is greater than 1 or because repeated convergence failures or error test failures occur. The computational cost can be significant if any difficulty is encountered because of the corrective actions—described later in this section—performed by the code. Even if the step is successful, the solution is not meaningful because of roundoff errors.

To avoid these difficulties, the calculation procedure for TOLSF uses $\underline{Y}_{n-1}$, which is known, so that the test can be applied at the start of each step, including the first. Thus the code ascertains inexpensively if excessive accuracy has been requested before attempting to advance the solution by the next integration step. The value of TOLSF may be used to adjust the local error tolerances so that this condition does not recur. For example, scaling up the $\{\mathrm{RTOL}_i\}$ and $\{\mathrm{ATOL}_i\}$ values by a minimum factor of TOLSF should produce satisfactory values for the local error tolerances if the same type of error control is to be performed (see chapter 4 for details).

3.4.4 Calculation of Method Coefficients

The integration method coefficients and test constants used to check corrector convergence and local accuracy, as well as to select method order and step size, are computed in subroutine CFODE. The calculation procedure uses the generating polynomials discussed by Hindmarsh (refs. 21 and 22) to increase portability of the code. The coefficients corresponding to all method orders are computed and stored both at the start of the problem and whenever the user changes the integration method. This feature avoids the computational cost associated with recomputing these quantities whenever the method order is changed.

3.4.5 Numerical Jacobians

If Newton-Raphson (NR) or Jacobi-Newton (JN) iteration is selected, the code will generate elements of the Jacobian matrix by finite-difference approximations if the user chooses not to provide an analytical Jacobian. For the iteration procedures corresponding to MITER = 2 (full Jacobian matrix) and 5 ("banded" Jacobian matrix, i.e., a matrix with many zero entries and all nonzero elements concentrated near the main diagonal), the element J_{ij} (= $\partial f_i/\partial y_j$) at ξ_n is estimated by using the approximation

$$J_{ij} \cong \frac{f_i\left(\left\{Y_{k,n}^{[0]} + \delta_{kj}\Delta Y_j\right\}, \xi_n\right) - f_i\left(\left\{Y_{k,n}^{[0]}\right\}, \xi_n\right)}{\Delta Y_j}, \quad i = 1, \ldots, N, \tag{3.20}$$

where $Y_{k,n}^{[0]}$ is the kth component of $Y_n^{[0]}$, δ_{kj} is the Kronecker symbol,

$$\delta_{kj} = \begin{cases} 0, & k \neq j \\ 1, & k = j, \end{cases} \tag{3.21}$$

and the increment ΔY_j in the jth solution component is selected as follows: The standard choice for ΔY_j is

$$\Delta Y_j = \sqrt{u}\left|Y_{j,n}^{[0]}\right|. \tag{3.22}$$

This equation cannot be used if $Y_{j,n}^{[0]}$ is either equal to zero or very small. Therefore an alternative value, based on noise level, is deduced as follows: Now the error in each f_i due to roundoff is of order $u|f_i|$. Hence in replacing $\partial f_i/\partial y_j$ by the difference quotient, equation (3.20), the resulting element J_{ij} has an error of order $u|f_i|/r_j$, where for clarity in presentation we have replaced ΔY_j by r_j. Finally because the method coefficient β_0 (= ℓ_0) is of order unity (see tables 2.1 and 2.2), the error δP_{ij} in the element P_{ij} of the iteration matrix **P**, equation (2.25), is approximately

$$\delta P_{ij} \simeq |h|u\,|f_i|/r_j. \tag{3.23}$$

If we introduce the N-dimensional column vector $\underset{\sim}{s}$, with element s_j defined as

$$s_j = 1/r_j, \quad j = 1, \ldots, N, \tag{3.24}$$

the matrix $\delta\mathbf{P}$ containing the errors $\{\delta P_{ij}\}$ is given by

$$\delta\mathbf{P} = |h|\, u\, |\underline{f}|\underline{s}^T, \tag{3.25}$$

where $|\underline{f}|$ is an N-dimensional column vector containing the absolute values of the f_i ($i = 1,\dots,N$) and the superscript T indicates transpose. A suitable increment r_j is obtained by bounding $|\delta\mathbf{P}|$, as discussed next.

To be consistent with the corrector convergence test, equation (2.98), and the local error test, equation (2.91), we use the weighted rms norm, which for an arbitrary N-dimensional column vector $\underline{x}$ is given by

$$\|\underline{x}\| = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i}{\mathrm{EWT}_i}\right)^2}. \tag{3.26}$$

If we introduce the diagonal matrix $\mathbf{D}$ of order N, with element D_{ii} given by

$$D_{ii} = 1/\mathrm{EWT}_i, \qquad i = 1,\dots,N, \tag{3.27}$$

it is easily verified that

$$\|\underline{x}\| = \|\mathbf{D}\underline{x}\|_E / \sqrt{N}, \tag{3.28}$$

where $\|\bullet\|_E$ is the Euclidean norm, defined for $\underline{x}$ as

$$\|\underline{x}\|_E = \sqrt{\sum_{i=1}^{N}|x_i|^2}. \tag{3.29}$$

Now the norm of $\delta\mathbf{P}$ is given by

$$\|\delta\mathbf{P}\| = \max_{\underline{x}} \frac{\|\delta\mathbf{P}\underline{x}\|}{\|\underline{x}\|}, \tag{3.30}$$

where

$$\|\delta\mathbf{P}\underline{x}\| = |h|\, u\, \||\underline{f}|\|\, |\underline{s}^T\underline{x}|, \tag{3.31}$$

because $\delta\mathbf{P}$ is of rank 1. Hence

$$\|\delta \mathbf{P}\| = |h|\, u\, \|\underline{f}\|\, \max_{\underline{x}} \frac{|\underline{s}^T \underline{x}|}{\|\underline{x}\|} = |h|\, u\, \|\underline{f}\|\, \max_{\underline{x}} \frac{\left|\left(\mathbf{D}^{-1}\underline{s}\right)^T \left(\mathbf{D}\underline{x}\right)\right|}{\|\underline{x}\|}$$

$$= |h|\, u\, \|\underline{f}\|\, \max_{\underline{x}} \frac{\|\mathbf{D}^{-1}\underline{s}\|_E \|\mathbf{D}\underline{x}\|_E}{\|\mathbf{D}\underline{x}\|_E / \sqrt{N}} = |h|\, u\, \|\underline{f}\| \sqrt{N}\, \|\mathbf{D}^{-1}\underline{s}\|_E$$

$$\leq |h|\, uN\, \|\underline{f}\|\, \max_i \left(\frac{\mathrm{EWT}_i}{r_i}\right),$$

which can be rewritten as

$$\|\delta \mathbf{P}\| \leq \frac{|h|\, uN\, \|\underline{f}\|}{\min_i \left(r_i / \mathrm{EWT}_i\right)}. \tag{3.32}$$

To establish the maximum allowable error in **P**, we consider the linear system $\mathbf{P}\underline{x} = \underline{b}$, which is the form of the equation to be solved at each Newton iteration, equation (2.24). To first order, the error $\delta\underline{x}$ in $\underline{x}$ due to the error $\delta\mathbf{P}$ in **P** is given by (e.g., ref. 13)

$$\frac{\|\delta \underline{x}\|}{\|\underline{x}\|} \leq \|\mathbf{P}^{-1}\|\, \|\delta \mathbf{P}\|. \tag{3.33}$$

The norm $\| \mathbf{P}^{-1} \|$ is not known but is expected to be of order unity because $\mathbf{P} \to \mathbf{I}$, the identity matrix of order N, when $h \to 0$ and $\mathbf{P} \sim -h\beta_0\mathbf{J}$ when $h \to \infty$ (see eq. (2.25)). Therefore, a reasonable strategy is to bound $\| \delta\mathbf{P} \|$ alone by selecting a suitably small value for the relative error that can be tolerated in the Newton correction vector. By using a value of 0.1 percent for this error, we obtain from equations (3.32) and (3.33)

$$\min\left(\frac{r_i}{\mathrm{EWT}_i}\right) \geq 10^3\, |h|\, uN\, \|\underline{f}\| \equiv r_0. \tag{3.34}$$

For additional safety r_0 is reset to 1 if it is equal to zero. Finally the increment ΔY_j in the jth variable used to estimate the $\{J_{ij}\}$ is given by

$$\Delta Y_j = \max\left(\sqrt{u}\left|Y_{j,n}^{[0]}\right|,\ r_0\ \mathrm{EWT}_{j,n}\right). \tag{3.35}$$

For a full Jacobian matrix the above procedure will require ($N + 1$) derivative evaluations and can therefore become much more expensive than the use of an analytical Jacobian, especially for large N. Now $\underline{f}(\underline{Y}_n^{[0]}, \xi_n)$ is required by the corrector (see eq. (2.36)), irrespective of the iteration technique. Hence the use of MITER = 2 requires the evaluation of only N additional derivatives.

In generating the finite-difference banded Jacobian matrix (MITER = 5) the code exploits the bandedness of the matrix for efficiency. The number of additional derivative evaluations required to form the Jacobian matrix is only ML + MU + 1, where ML and MU are, respectively, the lower and upper half-bandwidths of the Jacobian matrix.

If JN iteration with MITER = 3 is used, the N diagonal elements J_{ii} ($i = 1,...,N$) are estimated by using the approximation

$$J_{ii} \cong \frac{f_i\left(\underline{Y}_n^{[0]} + \Delta\underline{Y},\ \xi_n\right) - f_i\left(\underline{Y}_n^{[0]},\ \xi_n\right)}{\Delta Y_i}, \qquad i = 1,...,N, \tag{3.36}$$

which requires only one additional derivative evaluation. The increment ΔY_i is selected as follows: Now equation (2.17) shows that if functional iteration were used, the correction $\underline{Y}_n^{[1]} - \underline{Y}_n^{[0]}$ that would be obtained on the first iteration is equal to the quantity $\beta_0\,\underline{g}(\underline{Y}_n^{[0]})$, where the vector function $\underline{g}$ is given by equation (2.16). The increment vector $\Delta\underline{Y}$ is taken to be 10 percent of this correction:

$$\Delta Y_i = 0.1\ \beta_0 g_i\left(\underline{Y}_n^{[0]}\right), \quad i = 1,...,N. \tag{3.37}$$

Hence the diagonal matrix approximation, equation (3.36), resembles a directional derivative of $\underline{f}$ taken in the same direction as the correction vector above. Also, this approximation gives the correct Jacobian if it is a constant diagonal matrix. If the magnitude of ΔY_i is less than $0.1u\beta_0$ EWT$_i$, that is, if $\left|g_i\left(\underline{Y}_n^{[0]}\right)\right| < u\mathrm{EWT}_i$, J_{ii} is set equal to zero.

3.4.6 Solution of Linear System of Equations

If NR iteration is used for the problem, a linear system of the form $\mathbf{P}\underline{x} = \underline{b}$ must be solved for the correction vector $\underline{x}$ at each iteration (see eq. (2.24)). The linear algebra necessary to solve this equation is performed by the LU method (e.g., refs. 5 and 36), rather than by explicitly inverting the iteration matrix, which will require prohibitive amounts of computer time (ref. 13). In the LU method the iteration matrix is factored into the product of two triangular matrices **L** and **U**. Solving equation (2.24) then requires the fairly simple solution of two triangular linear systems in succession.

LSODE also includes special procedures for the LU-decomposition of the iteration matrix and the solution of equation (2.24) when the matrix is known to be banded. Compared to a full matrix, it is significantly less expensive to form a banded matrix, perform its LU-decomposition, and solve the linear system of equations (refs. 5, 25, 26, and 36). An important advantage of LU-decomposing a banded matrix over inverting it is that, besides being faster, the triangular factors **L** and **U** lie within nearly the same bands as the original matrix, whereas the inverse is a full matrix (ref. 36). This feature makes the computation of the correction vector significantly faster with the LU method than by premultiplying the right-hand side of equation (2.24) with the inverse of the matrix.

If MITER = 3 is used for the problem, the resulting iteration matrix is diagonal (see eq. (3.36)). Its inverse can therefore be obtained trivially and is used to compute the corrections.

3.4.7 Jacobian Matrix Update

The difficulty with Newton-Raphson iteration is the computational cost associated with forming the Jacobian matrix and the linear algebra required to solve for the correction vector at each iteration. However, as discussed in chapter 2, the iteration matrix need not be very accurate. This fact is exploited to reduce the computational work associated with linear algebra by not updating **P** at every iteration. For additional savings it is updated only when the iteration does not converge. Hence the iteration matrix is only accurate enough for the solution to converge, and the same matrix may be used over several steps. It is also updated if three or more error test failures occur on any step. Now **P** may be altered if the coefficient $h\beta_0$ is changed (see eq. (2.25)) because a new step size and/or method order is selected. In order to minimize convergence failures caused by an inaccurate **P**, the code updates **P** and performs its LU-decomposition (or inversion if MITER = 3) if $h\beta_0$ has changed by more than 30 percent since the last update of **P**. In addition, for MITER = 3, because $\mathbf{P}^{-1}$ can be generated inexpensively, it is first modified to account for any change in $h\beta_0$ since its last update, before the corrections are computed. The reevaluation and LU-decomposition or inversion are also done whenever the user changes any input parameter required by the code. Finally the same **P** is used for a maximum number of 20 steps, after which it is reevaluated and LU-decomposed or inverted.

3.4.8 Corrector Iteration Convergence and Corrective Actions

Irrespective of the solution method and the corrector iteration technique, the maximum number of corrector iterations attempted on any step is set equal to 3, based on experience that a larger number increases the computational cost without a corresponding increase in the probability of successful convergence (refs. 19, 21, 22, and 25). In addition to performing the convergence test, equation (2.99), at each iteration, STODE examines the value of the convergence rate c_m, equation (2.102). If c_m is greater than 1, the iteration is clearly not converging. STODE exploits this fact by abandoning the iteration if c_m is greater than 2 after the second iteration.

If convergence is not obtained because either (1) equation (2.99) is not satisfied after three iterations or (2) $c_m > 2$ after the second iteration, the following corrective actions are taken: For NR and JN iterations, if **P** is not current, it is updated at $\underline{y} = \underline{Y}_n^{[0]}$ and LU-decomposed or inverted, and the step is retried with the same step size. However, if either **P** is current or functional iteration is used, a counter of convergence failures on the current step is increased by 1, the step size is reduced by a factor of 4, and the solution is attempted with the new step size. The same corrective actions are taken in the event of a singular iteration matrix.

This procedure is repeated until either convergence is obtained or the integration is abandoned because either (1) 10 convergence failures have occurred or (2) the step size has been reduced below a user-supplied minimum value $h_{\min}$. In the event of an error exit the index of the component with largest magnitude in the weighted local error vector is returned to the subprogram calling LSODE.

3.4.9 Local Truncation Error Test and Corrective Actions

After successful convergence STODE performs the local truncation error test, equation (2.96). If the error test fails, the step size is reduced and/or the method order is reduced by 1 by using the procedures outlined in section 3.4.10, and the step is retried. After two consecutive failures the step size is reduced by at least a factor of 5, and the step is retried with either the same or a reduced order. After three or more failures it is assumed that the derivatives that have accumulated in the Nordsieck history matrix have errors of the wrong order. Therefore the first derivative is recomputed and the method order is set equal to 1 if it is greater than 1. Then the step size is reduced by a factor of 10, the iteration matrix is formed and either LU-decomposed or inverted, and the step is retried with a new $\mathbf{z}_{n-1}$ that is constructed from $\underline{Y}_{n-1}$ and $\dot{\underline{Y}}_{n-1} = \underline{f}(\underline{Y}_{n-1})$.

This procedure is repeated until either the error test is passed or an error exit is taken because either (1) 10 error test failures have occurred or (2) the step size has been reduced below $h_{\min}$. In the event of an error exit LSODE returns the index of the component with the largest magnitude in the weighted local error vector to the calling subprogram.

If the accuracy test is passed, the step is accepted as successful, and the Nordsieck history matrix $\mathbf{z}_n$ and the estimated local truncation error vector $\underline{d}_n$ at ξ_n are computed by using equations (2.76) and (2.89), respectively. Irrespective of whether the step was successful or not, STODE saves the value of the most recent step size attempted on the step so that the user may, if desired, change it.

3.4.10 Step Size and Method Order Selection

In addition to advancing the solution STODE periodically computes the method order and step size that together maximize efficiency while maintaining prescribed accuracy. As discussed in chapter 2, this result is accomplished by selecting the method order that maximizes step size. To simplify the algorithm, the code considers only the three method orders $q-1$, q, and $q+1$, where q is the current method order. For each method order the step size that will satisfy exactly the local error bound is computed by assuming that the highest derivative remains constant. The resulting step size ratios (defined as the ratio of the step size to be attempted on the next step to the current value h_n) are given by equations (2.107), (2.103), and (2.112), respectively, for method orders $q-1$, q, and $q+1$. These equations are, however, modified by using certain safety factors (1) to produce a smaller step size than the value that satisfies the error bound exactly, because the error estimates are not exact and the highest derivative is not usually constant, and (2) to bias the order-changing decision in favor of not changing the order at all, because any change in order requires additional work, and then in favor of decreasing the order, because an order reduction results in less work per subsequent step than an order increase. The formulas used in STODE to calculate the step size ratios are as follows:

$$r_{\text{down}} = \frac{1}{1.3\left[\left(D_{q-1}\right)^{\frac{1}{q}} + 10^{-6}\right]}, \tag{3.38}$$

$$r_{\text{same}} = \frac{1}{1.2\left[\left(D_{q}\right)^{\frac{1}{q+1}} + 10^{-6}\right]}, \tag{3.39}$$

$$r_{\text{up}} = \frac{1}{1.4\left[\left(D_{q+1}\right)^{\frac{1}{q+2}} + 10^{-6}\right]}. \tag{3.40}$$

In equations (3.38) to (3.40) the factors 1.2, 1.3, 1.4, and 10^{-6} are strictly empirical. The subscripts "down," "same," and "up" indicate, respectively, that the method order is to be reduced by 1, left unchanged, and increased by 1.

To prevent an order increase either after a failed step or when $q = q_{\max}$, the maximum order allowed for the solution method, r_{up} is set equal to zero in such cases. Similarly, if $q = 1$, r_{down} is set equal to zero to avoid an order reduction.

The maximum step size ratio $r = \max(r_{\mathrm{down}}, r_{\mathrm{same}}, r_{\mathrm{up}})$ and the corresponding method order are selected to be attempted on the next step if $r \geq 1.1$ after a successful step. Changes in both step size and method order are rejected if the step size increase is less than 10 percent because it is not considered large enough to justify the computational cost required by either change (refs. 10 and 22). After a failed step the method order is decreased if $r_{\mathrm{down}} > r_{\mathrm{same}}$; however, $r = \max(r_{\mathrm{down}}, r_{\mathrm{same}})$ is reset to 1 if it is greater than 1. Several additional tests, given next, are performed on r, if $r \geq 1.1$ after a successful step, but irrespective of the value of r after a failed step, before the step size h' $(= rh_n)$ to be attempted next is selected.

If the maximum step size $h_{\max}$ to be attempted on any step has been specified by the user, r is restricted to

$$r \leftarrow \min\left(r, \frac{h_{\max}}{h_n}\right). \tag{3.41}$$

Similarly if the user has specified a minimum step size $h_{\min}$ that may be attempted on any step, r is restricted to

$$r \leftarrow \max\left(r, \frac{h_{\min}}{h_n}\right). \tag{3.42}$$

Finally r must satisfy the inequality

$$r \leq r_{\max}, \tag{3.43}$$

where the variable $r_{\max}$ is normally set equal to 10. However, for the very first step size increase for the problem, if no convergence or error test failure has occurred, $r_{\max}$ is set equal to 10^4 to compensate for the small step size attempted on the first step. For the first step size increase following either a corrector convergence failure or a truncation error test failure, $r_{\max}$ is set equal to 2 to inhibit a recurrence of the failure.

To avoid numerical instability caused by frequent changes in the step size, method order and step size changes are attempted only after S successful steps with the same method order and step size, where S is normally set equal to $q + 1$. However, if an unsuccessful step occurs, this rule is disregarded and the step size and/or the method order may be reduced. Following a failed error test or a failed convergence test with either functional iteration or NR and JN iterations if **P** is current, S is set equal to $q + 1$. If three or more error test failures occur on any one step, S is set equal to 5 even though the method order is reduced to 1. Finally following a step for which step size and method order changes are rejected because $r < 1.1$, S is set equal to 3.

After every $S - 1$ successful steps STODE saves the vector $\underset{\sim}{e}$, if $q < q_{\max}$, in order to estimate $\nabla\underset{\sim}{e}$, which is required to compute r_{up} (see eqs. (2.109) to (2.112)). To minimize storage requirements, $\underset{\sim}{e}_n$ is saved as the $q_{\max}$th, that is, the last, column of $\mathbf{z}_n$.

3.5 Error Messages

The code contains many error messages—too numerous to list here. Every input parameter is tested for legality and consistency with the other input variables. If an illegal input parameter is discovered, a detailed message is printed. Each error message is self-explanatory and complete. It not only describes the mistake but in some instances tells the user how to fix the problem. Any difficulty encountered during execution will result in an error exit. A message giving the reason for termination will also be printed. If the computation stops prematurely, the user should look for the error message near the end of the output file corresponding to the logical unit number LUNIT (see chapter 4).

Chapter 4
Description of Code Usage

To use the LSODE package, the following subprograms must be provided: (1) a routine that manages the calls to subroutine LSODE, (2) a routine that computes the derivatives $\{f_i = dy_i/d\xi\}$ for given values of the independent variable ξ and the solution vector $\underline{y}$, and (3) if an analytical Jacobian matrix **J** (= $\partial\underline{f}/\partial\underline{y}$) is required by the corrector iteration technique selected by the user, a routine that computes the elements of this matrix. In addition, some modifications, discussed below, to the LSODE source itself may be necessary.

4.1 Code Installation

4.1.1 BLOCK DATA Variables

The user may wish to reset the values for the integer variables MESFLG (currently 1) and LUNIT (currently 6), which are both set either in the BLOCK DATA module (double-precision version) or in subroutine XERRWV (single-precision version). The variable MESFLG controls the printing of error messages from the code, and LUNIT is the logical unit number for such output (see table 3.7). Setting MESFLG = 0 will switch off all output from the code and therefore is not recommended.

The single-precision version of the code loads initial values for the common block LS0001 variables ILLIN and NTREP (see table 3.8) through a DATA statement in subroutine LSODE. The same procedure is used in subroutine XERRWV for the common block EH0001 variables MESFLG and LUNIT (see table 3.7). However, on some computer systems initial values for common block elements cannot be defined by means of DATA statements outside a BLOCK DATA subprogram. In this case the user must provide a separate BLOCK DATA subprogram, to which the two DATA statements from subroutines LSODE and XERRWV must be moved. The BLOCK DATA subprogram must also contain the two common blocks EH0001 and LS0001 (see table 3.6).

4.1.2 Modifying Subroutine XERRWV

The subroutine XERRWV, which prints error messages from the code, is machine and language dependent. Therefore the data type declaration for the argument MSG, which is a Hollerith literal or integer array containing the message to be printed, may have to be changed. The number of Hollerith characters stored per word is assumed to be 4, and the value of NMES, which is the length of, that is, number of characters in, MSG is assumed to be a multiple of 4, and at most 60. However, the routine describes the necessary modifications for several machine environments. In particular, the user must change a DATA statement and the format of statement number 10. The routine assumes that all errors are either (1) recoverable, in which case control returns to the calling subprogram, or (2) fatal, in which case the run is aborted by passing control to the statement STOP, which may be machine dependent. If a different run-abort command is needed, the line following statement number 100, which is located near the end of the routine, must be changed.

4.2 Call Sequence

The call sequence to subroutine LSODE is as follows:

CALL LSODE (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK, ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF)

All arguments in the call sequence are used on input, but only Y, T, ISTATE, RWORK, and IWORK are used on output. Also, Y and T are set only on the first call to LSODE; the other arguments may, however, have to be reset on subsequent calls. The arguments to LSODE are defined as follows:

F	The name of the user-supplied subroutine that computes the derivatives of the dependent variables with respect to the independent variable. This name must be declared EXTERNAL in the subprogram calling LSODE. The requirements of subroutine F are described in section 4.3.
NEQ	The number of first-order ordinary differential equations (ODE's) to be solved. (The code allows the user to decrease the value of NEQ during the course of solving the problem. This option is useful if some variables can be discarded as the solution evolves as, for example, in chemical kinetics problems for which the reaction mechanism is reduced dynamically.) As discussed later, NEQ can be specified as an array. In this case NEQ(1) must give the number of ODE's to be solved, and the subprogram calling LSODE must contain a dimension statement for NEQ.

Y	A vector of length NEQ (or more) containing the dependent variables. The subprogram calling LSODE must include a dimension statement for Y if it contains more than one component. On the first call to LSODE this vector must be set equal to the vector of initial values of the dependent variables. Upon every return from LSODE, Y is the solution vector either at the desired value (TOUT or TCRIT, see below) of the independent variable or that generated at the end of the previous integration step. In case of an error exit Y contains the solution at the last step successfully completed by the integrator.
T	The independent variable. On the first call to LSODE, T must give the initial value of this variable. On every return from LSODE, T is either the independent variable value (TOUT or TCRIT, see below) at which the solution is desired or the independent variable value to which the numerical solution was advanced on the previous integration step. If an error exit occurs, T gives the value of the farthest point (in the direction of integration) reached by the integrator.
TOUT	The next value of the independent variable at which the solution is required, if ITASK = 1, 3, or 4 (see table 4.1). For ITASK = 2 or 5, LSODE uses TOUT on the first call to determine the direction of integration and, if necessary, to compute the step size to be attempted on the first step; on subsequent calls TOUT is ignored. LSODE permits integration in either direction of the independent variable.
ITOL	A flag that indicates the type of local error control to be performed. The legal values that can be assigned for ITOL and their meanings are

TABLE 4.1.—VALUES OF ITASK USED IN LSODE AND THEIR MEANINGS

ITASK	Description
[a]1	Compute output values of $\underline{Y}(\xi)$ at $\xi = \xi_{out}$ by overshooting and interpolation.
2	Advance the solution to the ODE's by one step and return to calling subprogram.
[a]3	Stop at the first internal mesh point at or beyond $\xi = \xi_{out}$ and return to calling subprogram.
[a,b]4	Compute output values of $\underline{Y}(\xi)$ at $\xi = \xi_{out}$ but without overshooting $\xi = \xi_{crit}$.
[b]5	Advance the solution to the ODE's by one step without passing $\xi = \xi_{crit}$ and return to calling subprogram.

[a]User must supply value for ξ_{out} (= TOUT).
[b]User must supply value for ξ_{crit} (= TCRIT). This option is useful if the problem has a singularity at or beyond $\xi = \xi_{crit}$.

TABLE 4.2.—VALUES OF ITOL USED IN LSODE AND THEIR MEANINGS

ITOL	Description
1	Scalar RTOL and scalar ATOL
2	Scalar RTOL and array ATOL
3	Array RTOL and scalar ATOL
4	Array RTOL and array ATOL

given in table 4.2. The variables RTOL and ATOL are described next.

RTOL The local relative error tolerance parameter for the solution. This parameter can be specified either as a scalar, so that the same tolerance is used for all dependent variables, or as any array of length NEQ, so that different tolerances are used for different variables. In the latter case the subprogram calling LSODE must contain a dimension statement for RTOL.

ATOL The local absolute error tolerance parameter for the solution. This parameter can also be specified either as a scalar, so that the same tolerance is used for all dependent variables, or as an array of length NEQ, so that different tolerances are used for different variables. In the latter case the subprogram calling LSODE must contain a dimension statement for ATOL.

ITASK An index that specifies the task to be performed. This flag controls when LSODE stops the integration and returns the solution to the calling subprogram. The legal values for ITASK and their meanings are given in table 4.1. If ITASK = 4 or 5, the input variable TCRIT (= independent variable value that the integrator must not overshoot, see table 4.1) must be passed to LSODE as the first element of the array RWORK (defined below).

ISTATE An index that specifies the state of the calculation, that is, if the call to LSODE is the first one for the problem or if it is a continuation. The legal values for ISTATE that can be used on input and their meanings are given in table 4.3. The option ISTATE = 3 allows changes in the input parameters NEQ, ITOL, RTOL, ATOL, IOPT, MF, ML, and MU and any optional input parameter, except H0, discussed in the descriptions of RWORK and IWORK. The integer variables IOPT, MF, ML, and MU are defined below. The parameters ITOL, RTOL, and ATOL may also be changed with ISTATE = 2, but LSODE does not then check the legality of the new values. On return from LSODE, ISTATE has the values and meanings given in table 4.4.

TABLE 4.3.—VALUES OF ISTATE THAT CAN BE USED ON INPUT TO LSODE AND THEIR MEANINGS

ISTATE	Description
1	This is the first call for the problem.
2	This is not the first call for the problem, and the calculation is to be continued normally with no change in any input parameters except possibly ξ_{out} and ITASK.[a]
3	This is not the first call for the problem, and the calculation is to be continued normally, but with a change in input parameters other than ξ_{out} and ITASK.[a]

[a]See table 4.1 for description of ITASK.

TABLE 4.4.—VALUES OF ISTATE RETURNED BY LSODE AND THEIR MEANINGS

ISTATE	Meaning
1	Nothing was done because TOUT = T on first call to LSODE. (However, an internal counter was set to detect and prevent repeated calls of this type.)
2	The integration was performed successfully.
–1	Excessive amount of work was done on this call (i.e., number of steps exceeded MXSTEP[a] on this call), but the integration was successful as far as the value returned in T.
–2	Too much accuracy was requested for the computer being used, but the integration was successful as far as the value returned in T. (If this error is detected on the first call to LSODE (i.e., before any integration is done), an illegal input error (ISTATE = –3, see below) occurs instead.)
–3	Illegal input was specified. The error message is detailed and self-explanatory.
–4	Repeated error test failures occurred on one step, but the integration was successful as far as the value returned in T.
–5	Repeated convergence test failures occurred on one step, but the integration was successful as far as the value returned in T.
–6	Some component, EWT_i, of the error weight vector $\underline{EWT}$[b] vanished, so that the local error test cannot be applied, but the integration was successful as far as the value returned in T. (This condition arises when pure relative error control (i.e., $ATOL_i = 0$[b]) was specified for a variable whose magnitude is now zero.)

[a]See table 4.6.
[b]See chapter 2.

IOPT An integer flag that specifies if any optional input is being used on this call. The legal values for IOPT together with their meanings are given in table 4.5. The optional input parameters that may be set by the user are given in table 4.6. For each such input variable this table lists its location in the call sequence, its meaning, and its default value. The quantities RWORK and IWORK are work arrays described below.

TABLE 4.5.—VALUES OF IOPT THAT CAN BE USED ON INPUT TO LSODE AND THEIR MEANINGS

IOPT	Meaning
0	The user has not set a value for any optional input parameter.[a] (Default values will be used for all these parameters.)
1	Values have been specified for one or more optional input parameters.[a]

[a]See table 4.6 for a list of these parameters.

TABLE 4.6.—OPTIONAL INPUT PARAMETERS THAT CAN BE SET BY USER AND THEIR LOCATIONS, MEANINGS, AND DEFAULT VALUES

Optional input parameter	Location	Meaning	Default value
H0	RWORK(5)	Step size to be attempted on the first step	Computed by LSODE
HMAX	RWORK(6)	Absolute value of largest step size (in magnitude) to be used on any step	∞
HMIN	RWORK(7)	Absolute value of smallest step size (in magnitude) to be used on any step[a]	0
MAXORD	IWORK(5)	Maximum method order to be used on any step	12 for Adams-Moulton method and 5 for backward differentiation formula method
MXSTEP	IWORK(6)	Maximum number of integration steps allowed on any one call to LSODE	500
MXHNIL	IWORK(7)	Maximum number of times that warning message that step size is getting too small is printed	10

[a]This value is ignored on the first step and on the final step to reach TCRIT when ITASK = 4 or 5 (see table 4.1).

RWORK	A real work array used by the integrator. The subprogram calling LSODE must include a dimension statement for RWORK. If ITASK = 4 or 5, the user must set RWORK(1) = TCRIT (see table 4.1) to transmit this variable to LSODE. If any optional real input parameters are used, their values are also passed in this array to LSODE; the address for each of these parameters is given in table 4.6. Upon return from LSODE, RWORK contains several optional real output parameters. For each such output variable table 4.7 lists its location in RWORK and its meaning. In addition, the Nordsieck history array at the current value of the independent variable (TCUR in table 4.7) and the estimated local error vector in the solution incurred on the last successful step can be obtained from RWORK. Table 4.8 lists the names used for these two quantities and their locations in RWORK. In this table NYH is the value of NEQ on the first call to LSODE, and NQCUR and LENRW are both defined in table 4.7, which also gives their locations in the array IWORK (see below).
LRW	Length of the real work array RWORK. Its minimum value depends on the method flag MF (see below) and is given in table 4.9 for each legal value of MF. In this table the integer MAXORD is the maximum method order (default values = 12 and 5 for the AM and BDF methods, respectively) to be used. The integers ML and MU are the lower and upper half-bandwidths, respectively, of the Jacobian matrix if it is declared to be banded (see table 3.2).
IWORK	An integer work array used by the integrator. The subprogram calling LSODE must include a dimension statement for IWORK. If MITER (= second decimal digit of MF, defined below) = 4 or 5 (table 3.2), the user must set IWORK(1) = ML and IWORK(2) = MU (see descriptions above) to transmit these variables to LSODE. If any optional integer input parameters are used, their values are also passed in this array to LSODE; the address for each of these parameters is given in table 4.6. Upon return from LSODE, IWORK contains several optional integer output parameters. For each such output variable table 4.7 lists its location in IWORK and its meaning.
LIW	Length of the integer work array IWORK. Its minimum value depends on MITER (table 3.2) and is given in table 4.10 for each legal value of MITER.
JAC	The name of the user-supplied subroutine that computes the elements of the Jacobian matrix. This name must be declared EXTERNAL in the subprogram calling LSODE. The form and description of subroutine JAC are given in section 4.4.

TABLE 4.7.—OPTIONAL OUTPUT PARAMETERS RETURNED BY LSODE AND THEIR LOCATIONS AND MEANINGS

Optional output parameter	Location	Meaning
HU	RWORK(11)	Step size used on last successful step
HCUR	RWORK(12)	Step size to be attempted on next step
TCUR	RWORK(13)	Current value of independent variable. The integrator has successfully advanced the solution to this point.
TOLSF	RWORK(14)	A tolerance scale factor, greater than 1.0, that is computed when too much accuracy is requested (ISTATE = –2 or –3, see table 4.4). To continue integration with the same ITOL, the local error tolerance parameters RTOL and ATOL must both be increased by at least a factor of TOLSF.
NST	IWORK(11)	Number of integration steps used so far for problem
NFE	IWORK(12)	Number of derivative evaluations required so far for problem
NJE	IWORK(13)	Number of Jacobian matrix evaluations (and iteration matrix LU-decompositions or inversions) so far for problem
NQU	IWORK(14)	Method order used on last successful step
NQCUR	IWORK(15)	Method order to be attempted on next step
IMXER	IWORK(16)	Index of component with largest magnitude in weighted local error vector (e_i/EWT$_i$, see chapter 2). This quantity is computed when repeated convergence or local error test failures occur.
LENRW	IWORK(17)	Required length for array RWORK
LENIW	IWORK(18)	Required length for array IWORK

TABLE 4.8.—USEFUL INFORMATIONAL QUANTITIES REGARDING INTEGRATION THAT CAN BE OBTAINED FROM ARRAY RWORK AND THEIR NAMES AND LOCATIONS

Quantity	Name	Location
Nordsieck history array for problem	YH	RWORK(21) to RWORK(20 + NYH(NQCUR + 1))
Estimated local error in solution on last successful step	ACOR	RWORK(LENRW – NEQ + 1) to RWORK(LENRW)

TABLE 4.9.—MINIMUM LENGTH REQUIRED BY REAL WORK ARRAY RWORK (i.e., MINIMUM LRW) FOR EACH MF

MF	Minimum LRW[a]
10,20	20 + NYH(MAXORD + 1) + 3 NEQ
11,12,21,22	22 + NYH(MAXORD + 1) + 3 NEQ + $(NEQ)^2$
13,23	22 + NYH(MAXORD + 1) + 4 NEQ
14,15,24,25	22 + NYH(MAXORD + 1) + (2 ML + MU + 4)NEQ

[a]NYH is the number of ODE's specified on first call to LSODE, MAXORD is the maximum method order to be used for problem, NEQ is the number of ODE's specified on current call to LSODE, and ML and MU are, respectively, the lower and upper half-bandwidths of the banded Jacobian matrix.

TABLE 4.10.—MINIMUM LENGTH REQUIRED BY INTEGER WORK ARRAY IWORK (i.e., MINIMUM LIW) FOR EACH MITER

MITER[a]	Minimum LIW[b]
0	20
1,2	20 + NEQ
3	20
4,5	20 + NEQ

[a]See table 3.2 for description of MITER.
[b]NEQ is the number of ODE's specified on current call to LSODE.

MF Method flag that indicates both the integration method and corrector iteration technique to be used. MF consists of the two decimal digits METH, which specifies the integration method, and MITER, which specifies the iteration technique (eq. (3.1)). Equation (3.1) and tables 3.1 and 3.2 show that MF has the following 12 legal values—10, 11, 12, 13, 14, 15, 20, 21, 22, 23, 24, and 25. If MF = 14, 15, 24, or 25, the values of ML and MU must be passed to LSODE as the first and second elements, respectively, of the array IWORK (see above).

4.3 User-Supplied Subroutine for Derivatives (F)

Irrespective of the solution method or corrector iteration technique selected to solve the problem, the user must provide a subroutine that computes the derivatives $\{f_i\}$ for given values of the independent variable and the solution vector. The name (F) of this subroutine is an argument in the call vector to LSODE and must therefore be declared EXTERNAL in the subprogram calling LSODE. The derivative subroutine F should have the form

```
SUBROUTINE F (NEQ, T, Y, YDOT)
DIMENSION Y(1), YDOT(1) in FORTRAN 66
or DIMENSION Y(*), YDOT(*) in FORTRAN 77
```

In addition, if NEQ is an array, the subroutine F should include a DIMENSION statement for it. The routine F should not alter the values in T, NEQ (or NEQ(1), if NEQ is an array), or the first N elements in Y, where N is the current number of ODE's to be solved. The derivative vector should be returned in the array YDOT, with YDOT(I) = $dy_i/d\xi$ (i = I), evaluated at ξ = T, $\underline{y}$ = Y.

If the calculation of $\{f_i\}$ involves intermediate quantities whose current values, that is, at $\xi = \xi_n$ (or ξ_{out}), are required externally to LSODE, a special calculation, such as a call to the routine F, must be made. The results of the last call from the package to the routine F should not be used because they correspond to a Y value that is different from $\underline{Y}_n$ [or $\underline{Y}(\xi_{out})$] and a ξ value that may be different from ξ_n (or ξ_{out}). Here ξ_n is the independent variable value to which the numerical solution was advanced on the previous integration step and ξ_{out} = TOUT. If a special call to subroutine F is made, to reduce the storage requirement, the YDOT argument may be replaced with RWORK(LSAVF), the base address of an N-dimensional array, SAVF (see table 3.8), used for temporary storage by LSODE; LSAVF is the 224th word (6th integer word after 218 real words) in the common block LS0001 (table 3.6). If the derivative $\dot{\underline{Y}}_n$ is required, it can be obtained by calling subroutine INTDY, as explained in section 4.8.

4.4 User-Supplied Subroutine for Analytical Jacobian (JAC)

If the corrector iteration technique selected by the user requires a Jacobian matrix, we recommend that a routine that computes an analytical Jacobian be provided. The name (JAC) of this routine is an argument in the call vector to LSODE and must therefore be declared EXTERNAL in the subprogram calling LSODE. The Jacobian subroutine JAC should have the form

```
SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
DIMENSION Y(1), PD (NROWPD, 1) in FORTRAN 66
DIMENSION Y(*), PD (NROWPD, *) in FORTRAN 77
```

Here ML and MU are, respectively, the (user-supplied) lower and upper half-bandwidths of the Jacobian matrix if it is banded; and NROWPD, which is set by the code, is the number of rows of the Jacobian matrix PD. For a banded matrix NROWPD is equal to the extended bandwidth (= 2ML + MU + 1), and for a full matrix it is equal to the current number N of ODE's. If NEQ is an array, the subprogram JAC must include a DIMENSION statement for it.

This routine should not alter the values in NEQ (or NEQ(1), if NEQ is an array), T, ML, MU, or NROWPD. However, the Y array may, if necessary, be altered. For a full Jacobian matrix (MITER = 1) the element PD(I,J) (I = 1,...,N; J = 1,...,N) must be loaded with $\partial f_i/\partial y_j\big|_{\xi=\mathrm{T};\underline{y}=\underline{\mathrm{Y}}}\left(i=\mathrm{I}; j=\mathrm{J}\right)$. In this case the arguments ML and MU are not needed. If the Jacobian matrix is banded (MITER = 4), the element $\partial f_i/\partial y_j$ (i = 1,...,N; i – ML $\le j \le i$ + MU) must be loaded into PD (I – J + MU + 1, J) (I = i; J = j). Thus each band of the Jacobian matrix must be loaded in column-wise manner, with diagonal lines of **J**, from the top down, loaded into the rows of PD. For a diagonal matrix ML = MU = 0, and the diagonal elements must be loaded into a single row of length N. In any case the solver sets all elements of PD equal to zero before calling JAC, so that only the nonzero elements need to be loaded. Also each call to subroutine JAC is preceded by a call to subroutine F with the same arguments NEQ, T, and Y. To improve computational efficiency, intermediate quantities needed by both routines may be saved by routine F in a common block, thereby avoiding recomputation by routine JAC. If necessary, even the derivatives at T can be accessed by JAC by means of this method.

If functional iteration (MITER = 0) or an internally generated Jacobian matrix (MITER = 2, 3, or 5) is used, a dummy version of JAC may nonetheless be required to satisfy the loader. This version may be given simply as follows:

```
SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
RETURN
END
```

4.5 Detailed Usage Notes

It is apparent from the description of the call sequence to LSODE that the code has many capabilities and therefore requires the user to set values for several parameters. To further clarify code usage and assist in selecting values for user-set parameters, we provide here a somewhat detailed guide. We first summarize how we expect the code to be normally used and then give detailed usage notes. Additional insight into code usage can be obtained from the discussions by Byrne and Hindmarsh (ref. 17), who examined in some detail the solution of 10 example problems representing a variety of problem types, and by Radhakrishnan (ref. 37), who studied the effects of various user-set parameters on the solution of stiff ODE's arising in combustion chemistry.

4.5.1 Normal Usage Mode

The normal mode of communication with LSODE may be summarized as follows:

(1) Set initial values in Y.
(2) Set NEQ, T, ITOL, RTOL, ATOL, LRW, LIW, and MF.
(3) Set TOUT = first output station, ITASK = 1, ISTATE = 1, and IOPT = 0.
(4) Call LSODE.
(5) Exit if ISTATE < 0.
(6) Do desired output of Y.
(7) Exit if problem is finished.
(8) Reset TOUT to next print station and return to step (4).

This procedure will result in LSODE (a) computing the step size to be attempted on the first step, (b) continuing the integration with step sizes generated internally until the first internal mesh point at or, more usually, just beyond TOUT, and (c) computing the solution at TOUT by interpolation. The returned value T will be set equal to TOUT exactly, and Y will contain the solution at TOUT. Because the normal output value of ISTATE is 2, it does not have to be reset for normal continuation.

4.5.2 Use of Other Options

The calling subprogram may also make use of other options included in the package. For example, in step (8) ISTATE could be reset to 3 to indicate that at TOUT some parameters, such as NEQ or MF, have been changed. The task to be performed, indicated by the value of ITASK, can, however, be changed without resetting ISTATE. In the event of integration difficulties parameter values may also be changed in step (5), followed by a return to step (4), if the new values will prevent a recurrence of the indicated trouble.

4.5.3 Dimensioning Variables

Irrespective of the options selected, the subprogram calling LSODE must include DIMENSION statements for all call sequence variables that are arrays. Such variables include Y, RTOL, ATOL, RWORK, IWORK, and, as discussed below, possibly NEQ. The solution vector Y may be declared to be of length NEQ or greater. The first NEQ elements of the Y array must be the variables whose ODE's are to be solved. The remaining locations, if any, may be used to store other real data to be passed to the routines F and/or JAC. The LSODE package accesses only the first NEQ elements of Y; the remaining elements are unchanged by the code.

The parameter NEQ is usually a scalar quantity. However, an array NEQ may be used to store and pass integer data to the routines F and/or JAC. In this case the first element of NEQ must be set equal to the number of ODE's. The LSODE package accesses only NEQ(1). However, NEQ is used as an argument in the calls to the routines F and JAC, so that these routines, and the MAIN program, must include NEQ in a DIMENSION statement.

4.5.4 Decreasing the Number of Differential Equations (NEQ)

In the course of solving a problem the user may decrease (but not increase) the number of ODE's. This option is useful if some variables reach steady-state values while others are still varying. Dropping these constant quantities from the ODE list decreases the size of the system and hence increases computational efficiency. To use this option, upon return from LSODE at the appropriate time, the calling subprogram must reset the value of NEQ (or NEQ(1)); set ISTATE = 3; reset the values of all other parameters that are either required to continue the integration, such as TOUT if ITASK = 1, 3, or 4 (table 4.1), or are changed at the user's option; and then call LSODE again. If the Jacobian matrix is declared to be banded (MITER = 4 or 5, table 3.2) and reductions can be made to the half-bandwidths ML and MU, they will also produce efficiency increases. The option of decreasing the number of ODE's may be exercised as often as the user wishes. Of course, each time the size of the ODE system is decreased the changes discussed above should be made and the resulting number of ODE's can never be less than 1. However, the LRW and LIW values need not be reset.

If, at any time, the number of ODE's is decreased from N to N′, LSODE will drop the last N – N′ ODE's from the system and integrate the first N′ equations. It is therefore important in formulating the problem to order the variables carefully and make sure that it is indeed the last N – N′ variables that attain steady-state values. In continuing the integration LSODE will access only the first N′ elements of Y. However, the remaining N – N′, or more, elements can be accessed by the user, and so no special programming is needed in either routine F or JAC.

4.5.5 Specification of Output Station (TOUT)

The argument TOUT must be reset every time LSODE is called if the option given by ITASK = 1, 3, or 4 is selected. For the other two values of ITASK (i.e., 2 and 5), TOUT need be set only on the first call to LSODE. Irrespective of the value of ITASK, the TOUT value provided on the first call to LSODE is used to determine the direction of integration and, if the user has not supplied a value for it, to compute the step size to be attempted on the first step. Therefore unless the user specifies the value for the initial step size, it is recommended that some thought be given to the value used for TOUT on the first call to LSODE.

On the first call to LSODE, that is, with ISTATE = 1, TOUT may be set equal to the initial value of the independent variable. In this case LSODE will do nothing, and so the value ISTATE = 1 will be returned to the calling subprogram; however, an internal counter will be updated to prevent repeated calls of this nature. If such a "first" call is made more than four times in a row, an error message will be issued and the execution terminated.

On the second and subsequent calls to LSODE there is no requirement that the TOUT values be monotonic. However, a value for TOUT that "backs up" is limited to the current internal interval [(TCUR – HU),TCUR], where TCUR is the current value of the independent variable and HU is the step size used on the previous step.

4.5.6 Specification of Critical Stopping Point (TCRIT)

In addition to TOUT a value must be specified for TCRIT if the option ITASK = 4 is selected. TCRIT may be equal to TOUT or beyond it, but not behind it, in the direction of integration. The integration is not permitted to overshoot TCRIT, so that the option is useful if, for example, a singularity exists at or beyond TCRIT. This variable is also required with the option ITASK = 5. In either case the first element of the array RWORK (i.e., RWORK(1)) must be set equal to TCRIT. If the solver reaches TCRIT within roundoff, it will return T = TCRIT exactly and the solution at TCRIT is returned in Y. To continue integrating beyond TCRIT, the user must reset either ITASK or TCRIT. In either case the value of ISTATE need not be reset. However, whenever TCRIT is changed, the new value must be loaded into RWORK(1).

4.5.7 Selection of Local Error Control Parameters (ITOL, RTOL, and ATOL)

Careful thought should be given to the choice of ITOL, which together with RTOL and ATOL determines the nature of the error control performed by LSODE. The value of ITOL dictates the value of the local error weight vector <u>EWT</u>, with element EWT_i defined as

$$\mathrm{EWT}_i = \mathrm{RTOL}_i |Y_i| + \mathrm{ATOL}_i, \tag{4.1}$$

where RTOL_i and ATOL_i are, respectively, the local relative and absolute error tolerances for the ith solution component Y_i and the bars $|\bullet|$ denote absolute value. The solver controls the estimated local errors $\{d_i\}$ in $\{Y_i\}$ by requiring the root-mean-square (rms) norm of d_i/EWT_i to be 1 or less.

Pure relative error control for the ith solution component is obtained by setting $\mathrm{ATOL}_i = 0$; RTOL_i is then a measure of the number of accurate significant figures in the numerical solution. This error control is generally appropriate when widely varying orders of magnitude in Y_i are expected. However, it cannot be used if the solution vanishes because relative error is then undefined. Pure absolute error control for the ith solution component is obtained by setting

$RTOL_i = 0$; $ATOL_i$ is then a measure of the largest number that may be neglected.

Both RTOL and ATOL can be specified (1) as scalars, so that the same error tolerances are used for all variables, or (2) as arrays, so that different tolerances are used for different variables. The value of the user-supplied parameter ITOL indicates whether RTOL and ATOL are scalars or arrays. The legal values that can be assigned to ITOL and the corresponding types of RTOL and ATOL are given in table 4.2. If RTOL and/or ATOL are arrays, the calling subprogram must include an appropriate DIMENSION statement. A scalar RTOL is generally appropriate if the same number of significant figures is acceptable for all components of Y. A scalar ATOL is generally appropriate when all components of Y, or at least their peak values, are expected to be of the same magnitude.

In addition to ITOL, RTOL and ATOL should be selected with care. Now the code controls an estimate of only the local error, that is, an estimate of the error committed on taking a single step, starting with data regarded as exact. However, what is of interest to the user is the global truncation error or the actual deviation of the numerical solution from the exact solution. This error accumulates in a nontrivial manner from the local errors and is neither measured nor controlled by the code. It is therefore recommended that the user be conservative in choosing values for the local error tolerance parameters. However, requesting too much accuracy for the precision of the machine will result in an error exit (table 4.4). In such an event the minimum factor TOLSF by which RTOL and ATOL should both be scaled up is returned by LSODE (see table 4.7). Some experimentation may be necessary to optimize the tolerance parameters, that is, to determine values that produce sufficiently accurate solutions while minimizing the execution time. The global errors in solutions generated with particular values for the local error tolerance parameters can be estimated by comparing them with results produced with smaller tolerances. In reducing the tolerances all components of RTOL and ATOL, and hence of $\underline{EWT}$, should be scaled down uniformly.

There is no requirement that the same values for ITOL, RTOL, and ATOL be used throughout the problem. If during the course of the problem any of these parameters is changed, the user should reset ISTATE = 3 before calling LSODE again. (ISTATE need not be reset; however, LSODE will not then check the legality of the new values.) This option is useful, for example, if the solution displays rapid changes in a small subinterval but is relatively smooth elsewhere. To accurately track the solution in the rapidly varying region, small values of RTOL and ATOL may be required. However, in the smooth regions these tolerances could be increased to minimize execution time.

4.5.8 Selection of Integration and Corrector Iteration Methods (MF)

The choice of the method flag MF may also require some experimentation. The user should consider the nature of the problem and storage requirements. The primary consideration regarding MF is stiffness. If the problem is not stiff, the best choice is probably MF = 10 (Adams-Moulton (AM) method with functional iteration.) If the problem is stiff to a significant degree, METH should be set equal to 2 (table 3.1), and MITER (table 3.2) depends on the structure of the Jacobian matrix. If the Jacobian is banded, MITER = 4 (user-supplied analytical Jacobian) or 5 (internally generated Jacobian by finite-difference approximations) should be used. For either of these two MITER values the user must set values for the lower (ML) and upper (MU) half-bandwidths of the Jacobian matrix. The first and second elements of the integer work array IWORK must be set equal to ML and MU, respectively; that is, IWORK(1) = ML and IWORK(2) = MU. For a full matrix MITER should be set equal to 1 (analytical Jacobian) or 2 (internally generated Jacobian). If the matrix is significantly diagonally dominant, the choice MITER = 3, that is, Jacobi-Newton (JN) iteration using an internally generated diagonal approximation for the Jacobian matrix, can be made. To use this iteration technique with an analytical Jacobian, set MITER = 4 and ML = MU = 0.

If the problem is only mildly stiff, the choice METH = 1 (i.e., the AM method) may be more efficient than METH = 2 (i.e., the backward differentiation formula (BDF) method). For this case experimentation would be necessary to identify the optimal METH. If the user has no a priori knowledge regarding the stiffness of the problem, one way to determine its nature is to try MF = 10 and examine the behavior of both the solution and step size pattern. (It is recommended that some upper limit be set for the total number of steps or derivative evaluations to avoid excessive run times.) If the typical values of the step size are much smaller than the solution behavior would appear to require, for example, more than 100 steps are taken over an interval in which the solution changes by less than 1 percent, the problem is probably stiff. The degree of stiffness can be estimated from the step sizes used and the smoothness of the solution.

Irrespective of the integration method selected, the least effective iteration technique is functional iteration, given by MITER = 0, and the most effective is Newton-Raphson (NR), given by MITER = 1 or 2 (4 or 5 for a banded Jacobian matrix). Generally JN iteration is somewhere in between. However, storage requirements increase in the same order as the effectiveness of the iteration technique (see table 4.9), and so trade-off considerations are necessary. For reasons of computational efficiency the user is encouraged to provide a routine for computing the analytical Jacobian, unless the system is fairly complicated and analytical expressions cannot be derived for the matrix elements. The accuracy of the Jacobian calculation can be checked by comparison with the **J** internally generated with MITER = 2 or 5. Jacobi-Newton iteration requires considerably less storage and execution time per iteration but will be effective only if the Jacobian matrix is significantly diagonally dominant.

The importance of supplying an analytical Jacobian matrix, especially for large problems, is illustrated by Radhakrishnan (ref. 37), who studied 12 test problems from combustion kinetics. The problems covered a wide range of reaction conditions and reaction mechanism size. The effects on solution efficiency of (1) METH, (2) the first output station, and (3) optimizing the local error tolerances were also examined.

4.5.9 Switching Integration and Corrector Iteration Methods

The user may specify different values for MF in different subintervals of the problem. This option is useful if the problem changes character and is nonstiff in some regions and stiff elsewhere. Because stiff problems are usually characterized by a nonstiff initial "transient" region, one could use MF = 10 in the initial region and then switch to MF = 21 (the BDF method with NR iteration using an analytical Jacobian matrix) in the later stiff regime. It is very straightforward to change integration methods and corrector iteration techniques. Upon return from LSODE the user simply resets MF to the desired new value. The other action required is to reset ISTATE = 3 before calling LSODE again. The lengths LRW and LIW, respectively, of the arrays RWORK and IWORK depend on MF (see tables 4.9 and 4.10). If different methods are to be used in the course of solving a problem, storage corresponding to at least the maximum values of LRW and LIW must be allocated. That is, the dimensions of RWORK and IWORK must be set equal to at least the largest of the LRW and LIW values, respectively, required by the different methods to be used.

4.6 Optional Input

In addition to the input parameters whose values are required by the code, the user can set values for several other parameters to control both the integration and the output from the code. These optional input parameters are given in table 4.6, together with their locations and default values. If any of these parameters are used, the user must set IOPT = 1 to relay this information to the solver, which will examine all optional input parameters and select only those for which nonzero values are specified. A value of zero for any parameter will cause its default value to be used. Thus to use a subset of the optional inputs, set RWORK(I) = 0.0 and IWORK(I) = 0 (I = 5 to 7), and then set parameters of interest to the desired (nonzero) values. The variable H0, the step size to be attempted on the first step, must indicate the direction of integration. That is, H0 must be a positive quantity for integration in the forward direction (increasing values of the independent variable) and negative otherwise. All other input parameters must be positive numbers; otherwise, an error exit will occur.

To reset any optional input parameter on a subsequent call to LSODE, ISTATE must be set equal to 3. IOPT is not altered by LSODE and therefore need not be reset. Also because the code does not alter the values in RWORK (5) to RWORK (7) and IWORK(5) to IWORK(7), only parameters for which new values are required need to be reset. To specify a default value for any parameter for which a nondefault value had previously been used, simply load the appropriate location in RWORK or IWORK with a zero. Of course, if all variables are to have default values, simply reset IOPT = 0.

4.6.1 Initial Step Size (H0)

The sign of the step size H0 must agree with the direction of integration; otherwise, an error exit will occur. Also, its magnitude should be considerably smaller than the average value expected for the problem because the code starts the integration with a first-order method. Of course, the integrator tests that the given step size does produce a solution that satisfies the local error test and, if necessary, decreases it (in magnitude). The only test made on the magnitude of H0 prior to taking the first step is that it does not exceed the user-supplied value for HMAX, the maximum absolute step size allowed for the problem.

4.6.2 Maximum Step Size (HMAX)

The user may have to specify a finite value for HMAX (default value, ∞) if the solution is characterized by rapidly varying transients between long smooth regions. If the step size is too large, the solver may skip over the fine detail that the user may be (primarily) interested in. An example of this behavior is the buildup of ozone and oxygen atom concentrations in the presence of sunlight (ref. 17).

4.6.3 Maximum Method Order (MAXORD)

The optional input parameter MAXORD, the maximum method order to be attempted on any step, should not exceed the default value—12 for the AM method and 5 for the BDF method. If it does, it will be reduced to the default value. Also, in the course of solving the problem, if MAXORD is decreased to a value less than the current method order, the latter quantity will be reduced to the new MAXORD.

The maximum method order has to be restricted to a value less than the default value for stiff problems when the eigenvalues of the Jacobian matrix are close to the imaginary axis; that is, the solution is highly oscillatory. In such a situation the BDF method of high order (≥ 3) has poor stability characteristics and, as the stability plots in Gear (ref. 10) show, the unstable region grows as the order is increased. For this reason MAXORD should be set equal to 3 unless the eigenvalues are imaginary; that is, $\mathrm{Re}(\lambda_i) = 0$ and $\mathrm{Im}(\lambda_i) \neq 0$, where $\mathrm{Re}(\lambda_i)$ and $\mathrm{Im}(\lambda_i)$ are the real and imaginary parts of λ_i, the ith eigenvalue. In this case the value MAXORD = 2 should be used.

4.7 Optional Output

The user is usually primarily interested in the numerical solution and the corresponding value of the independent variable. These quantities are always returned in the call variables Y and T. In addition, several optional output quantities that contain information about the integration are returned by LSODE. These quantitites are given in tables 4.7 and 4.8, together with their locations. Some of these quantities give a measure of the computational work required and may, for example, help the user decide if the problem is stiff or if the right method is being used. Other output quantities will, in the event of an error exit, help the user either set legal values for some parameters or identify the reason for repeated convergence failures or local error test failures.

4.8 Other Routines

To gain additional capabilities, the user can access the following subroutines included in the LSODE package: INTDY, SRCOM, XSETF, and XSETUN. Among these, only INTDY is used by LSODE.

4.8.1 Interpolation Routine (Subroutine INTDY)

The subroutine INTDY provides derivatives of Y, up to the current order, at a specified point T and may be called only after a successful return from LSODE. The call to this routine takes the form

CALL INTDY (T, K, RWORK(21), NYH, DKY, IFLAG)

where T, K, RWORK(21), and NYH are input parameters and DKY and IFLAG are output parameters. The arguments to INTDY are defined as follows:

T	Value of independent variable at which the results are required. For the results to be valid T must lie in the interval [(TCUR − HU),TCUR], where TCUR and HU are defined in table 4.7.
K	Integer that specifies the desired derivative order and must satisfy $0 \le K \le$ current method order NQCUR (see table 4.7 for location of this quantity). Now, because the method order is never less than 1, the first derivative $d\underline{Y}/d\xi$ can always be obtained by calling INTDY.
RWORK(21)	Base address of the Nordsieck history array (see table 4.8).
NYH	Number of ODE's used on the first call to LSODE. If the number of ODE's is decreased during the course of the problem, NYH should be saved. An alternative way of obtaining NYH is to include the common block LS0001 in the subprogram calling INTDY. LSODE saves NYH in LS0001 as the 232nd word—the 14th integer word after 218 real words (see table 3.6).

DKY	Array of length N that contains the Kth derivative of Y at T. The subprogram calling INTDY must include a DIMENSION statement for DKY if NYH > 1. Alternatively, to save storage, DKY can be replaced with RWORK(LSAVF)—see section 4.3.
IFLAG	An error flag with following values and meanings: 0 Both T and K were legal. –1 Illegal value was specified for K. –2 Illegal value was specified for T.

4.8.2 Using Restart Capability (Subroutine SRCOM)

The subroutine SRCOM is useful if one is either alternating between two or more problems being solved by LSODE or interested in interrupting a run and restarting it later. The latter situation may arise, for example, if one is interested in steady-state values with no a priori knowledge of the required integration interval. The run may be stopped periodically, the results examined and, if necessary, the integration continued. This procedure is clearly more economical than making repeated runs on the same problem with, say, increasing values of TOUT. To exploit the capability of stopping and then continuing the integration, the user must save and then restore the contents of the common blocks LS0001 and EH0001. This information can be stored and restored by calling SRCOM. The call to this routine takes the form

```
CALL SRCOM (RSAV, ISAV, JOB)
```

where RSAV must be declared as a real array of length 218 or more in the calling subprogram and ISAV as an integer array of length 41 or more and JOB is an integer flag whose value (= 1 or 2) indicates the action to be performed by SRCOM as follows: JOB = 1 means "save the contents of the two common blocks," and JOB = 2 means "restore this information."

Thus to store the contents of EH0001 and LS0001, SRCOM should be called as follows:

```
CALL SRCOM (RSAV, ISAV, 1)
```

Upon return from SRCOM, RSAV and ISAV will contain, respectively, the 218 real and 39 integer words that together make up the common block LS0001. The 40th and 41st elements of ISAV will contain the two integer words MESFLG and LUNIT in the common block EH0001 (table 3.6). The lengths and contents of the arrays RWORK and IWORK must also be saved. The lengths LENRW and LENIW required for the arrays RWORK and IWORK are saved by LSODE as the 17th and 18th elements, respectively, of the array IWORK (see table 4.7).

To continue the integration, the arrays RWORK and IWORK and the contents of the common blocks LS0001 and EH0001 must be restored. The common block contents are restored by using the previously saved arrays RSAV and ISAV and calling the routine SRCOM as follows:

CALL SRCOM (RSAV, ISAV, 2)

The user should then set values for the input parameters required by LSODE, and the integration can be continued by calling this routine. Note, in particular, that ISTATE must be set equal to 2 or 3 to inform LSODE that the present call is a continuation one for the problem (see table 4.3).

4.8.3 Error Message Control (Subroutines XSETF and XSETUN)

To reset the value of the logical unit number LUNIT for output of messages from the code, the routine XSETUN should be called as follows:

CALL XSETUN (LUN)

where LUN is the new value for LUNIT. Action is taken only if the specified value is greater than zero.

The value of the flag MESFLG, which controls whether messages from the code are printed or not, may be reset by calling subroutine XSETF as follows:

CALL XSETF (MFLAG)

where MFLAG is the new value for MESFLG. The legal values for MFLAG are 0 and 1. Specifying any other value will result in no change to the current value of MESFLG. Setting MFLAG = 0 does carry the risk of losing valuable information through error messages from the integrator.

4.9 Optionally Replaceable Routines

If none of the error control options included in the code are suitable, more general error controls can be obtained by substituting user-supplied versions of the routines EWSET and/or VNORM (table 3.3). Both routines are concerned with measuring the local error. Hence any replacement may have a major impact on the performance of the code. We therefore recommend that modifications be made only if absolutely necessary, and that too with great caution. Also the effect of the changes and the accuracy of the programming should be studied on some simple problems.

4.9.1 Setting Error Weights (Subroutine EWSET)

The subroutine EWSET sets the array of error weights EWT, equation (4.1). This routine takes the form

```
SUBROUTINE EWSET (N, ITOL, RTOL, ATOL, YH, EWT)
```

where N is the current value of the number of ODE's; ITOL, RTOL, ATOL, and EWT have been defined previously; and YH contains the current Nordsieck history array, that is, the current solution vector YCUR and its NQ scaled derivatives, where NQ is the current method order. On the first call to EWSET from the routine LSODE, YCUR is the same as the Y array (which then contains the initial values supplied by the user); thereafter the two arrays may be different.

The error weights {EWT_i} are used in the local truncation error test, which requires that the rms norm of d_i/EWT_i be 1 or less. Here, d_i is the estimated local error in Y_i. The above norm is computed in the routine VNORM (discussed in section 4.9.2) to which the EWT array is passed.

If the user replaces the current version of EWSET, the new version must return in each EWT_i (i = 1,...,N) a positive quantity for comparison with d_i. This routine is called by the routine LSODE only (tables 3.4 and 3.5). However, in addition to its use in the local truncation error test (which is performed in the routine STODE), EWT is used (1) by the routine LSODE in computing the initial step size H0 and the optional output integer IMXER (table 4.7) and (2) by the routine PREPJ in computing the increments in solution vector for the difference quotient Jacobian matrix (MITER = 2 or 5, table 3.2) and for the diagonal approximation to the Jacobian matrix (MITER = 3). The base address for EWT in the array RWORK is LEWT, which is the 222nd word (the 4th integer word after 218 real words) in the common block LS0001.

If the user's version of EWSET uses current values of the derivatives of $\underline{Y}$, they can be obtained from YH, as described later. Indeed, derivatives of any order, up to NQ, can be found from YH, whose base address in RWORK is LYH (= 21), the 221st word (the 3rd integer word past 218 real words) in LS0001. The array YH is of length NYH(NQ + 1), where NYH is the value of N on the first call to LSODE. The first N elements correspond exactly to the YCUR array. The remaining terms contain scaled derivatives of YCUR. For example, the N elements J•NYH + 1 to J•NYH + N (J = 0,1,...,NQ) contain the Jth scaled derivative $H^J\underline{Y}^{(J)}/J!$, where H is the current value of the step size. On the first call to EWSET, before any integration is done, H is (temporarily) set equal to 1.0. Thereafter its value may be determined from LS0001, where it is the 212th real word. This common block also contains NYH as the 232nd word (the 14th integer word past 218 real words) and NQ as the 253rd word (the 35th integer word past 218 real words). Thus if the user wishes to use the Jth derivative in EWSET, it may be obtained by including the following statements:

```
SUBROUTINE EWSET (N, ..., YH, ..., EWT)
REAL (or DOUBLE PRECISION) YH, EWT, RLS, H, ...
INTEGER N, ILS, NQ, NYH, ...
DIMENSION YH(1), EWT(1), ... in FORTRAN 66
DIMENSION YH(*), EWT(*), ... in FORTRAN 77
```

```
COMMON/LS0001/RLS(218), ILS(39)
NQ = ILS(35)
NYH = ILS(14)
H = RLS(212)
```

The Jth derivative ($0 \le J \le NQ$) is then given by

$$Y_I^{(J)} = \frac{J!\ YH(J*NYH+I)}{H^J}, \quad I = 1,\ldots,N, \tag{4.2}$$

where $Y_I^{(J)}$ is the Jth derivative of Y_I. The routine must include a data type declaration and a DIMENSION statement for $\underline{Y}^{(J)}$. To save on storage, these values may be stored temporarily in the vector EWT.

4.9.2 Vector-Norm Computation (Function VNORM)

The real (or double precision) function routine VNORM computes the weighted root-mean-square (rms) norm of a vector. It is used as follows:

D = VNORM (N, V, W)

where N is the length of the real arrays V, which contains the vector, and W, which contains the weights. Upon return from VNORM, D contains the weighted rms-norm

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{V_i}{W_i}\right)^2}.$$

This routine is used by STODE to compute the weighted rms norm of the estimated local error. STODE also uses information returned by VNORM to perform the corrector convergence test and to compute factors that determine if the method order should be changed. Other routines that access VNORM are LSODE, to compute the initial step size H0, and PREPJ, to compute the increments in the solution vector for generating difference quotient Jacobians (MITER = 2 or 5, table 3.2).

If the user replaces the routine VNORM, the new version must return a positive quantity in VNORM, suitable for use in local error and convergence testing. The weight array W can be used as needed, but it must not be altered in VNORM. For example, the max-norm, that is, $\max|V_i/W_i|$, satisfies this requirement, as does a norm that ignores some components of V. The latter procedure has the effect of suppressing error control on the corresponding components of $\underline{Y}$.

4.10 Overlay Situation

If LSODE is to be used in an overlay situation, the user must declare the variables in the call sequence to LSODE and in the two internal common blocks LS0001 and EH0001 in the MAIN program to ensure that their contents are preserved. The common block LS0001 is of length 257 (218 real or double-precision words followed by 39 integer words), and EH0001 contains two integer words (see table 3.6).

4.11 Troubleshooting

In this section we present a brief discussion of the corrective actions that may be taken in case of difficulty with the code. If the execution is terminated prematurely, the user should examine the error message and the value of ISTATE returned by LSODE (table 4.4). We therefore recommend that the current value of MESFLG not be changed, at least until the user has gained some experience with the code. The legality of every input parameter, both required and optional, is checked. If illegal input is detected by the code, it returns to the calling subprogram with ISTATE = –3. The error message will be detailed and will make clear what corrective actions to take. If the illegal input is caused by a request for too much accuracy, the user should examine the value of TOLSF returned in RWORK(13) (table 4.7) and make necessary adjustments to RTOL and ATOL, as described in section 4.5.7. If an excessive accuracy requirement is detected during the course of solving the problem, the value ISTATE = –2 is returned. To continue the integration, make the adjustments mentioned above, set ISTATE = 3, and call LSODE again.

Another difficulty related to accuracy control may be encountered if pure relative error control for, say, the ith variable is specified (i.e., $\mathrm{ATOL}_i = 0$). If this solution component vanishes, the error test cannot be applied. In this situation the value ISTATE = –6 is returned to the calling subprogram. The error message identifies the component causing the difficulty. To continue integrating, reset ATOL for this component to a nonzero value, set ISTATE = 3, and call LSODE again.

If more than MXSTEP (default value, 500) integration steps are taken on a single call to LSODE without completing the task, the error return ISTATE = –1 is made. The problem might be the use of an inappropriate integration method or iteration technique. The use of MF = 10 (or 20) on a stiff problem is one example. The user should, as described previously under the selection of MF (section 4.5.8), verify that the value of MF is right for the problem. Very stringent accuracy requirements may also cause this difficulty. Another possibility is that pure relative error control has been specified but most, or all, of the $|Y_i|$ are very small but nonzero. Finally, the solution may be varying very rapidly, forcing the integrator to select very small step sizes, or the integration interval may be very long relative to the average step size. To continue the integration, simply reset ISTATE = 2 and call LSODE again—the excess step counter will be reset to zero. To prevent a recurrence of the error, the value of MXSTEP can be increased, as described in section 4.6. If this action is taken between calls to LSODE, ISTATE must be set equal to 3 before LSODE is called again. Irrespective of when MXSTEP is increased, IOPT should be set equal to 1 before the next call to LSODE.

If the integrator encounters either repeated local error test failures or any local error test failure with a step size equal to the user-supplied minimum value HMIN (table 4.6), LSODE returns with ISTATE = –4. The difficulty could be caused by a singularity in the problem or by inappropriate input. The user should check subroutines F and JAC for errors. If none is found, it may be necessary to monitor intermediate quantities. The component IMXER causing the error test failure is returned as IWORK(16) (table 4.7). The values Y(IMXER), RTOL(IMXER), ATOL(IMXER), and ACOR(IMXER) (see table 4.8) should be examined. If pure relative error control had been specified for this component, very small but nonzero values of Y(IMXER) may cause the difficulty.

These checks should also be made if the integration fails because of either repeated corrector convergence test failures or any such failure with a step size equal to HMIN. In this case LSODE returns the value ISTATE = –5 along with a value for IMXER defined above. If an analytical Jacobian is being used, it should be checked for errors. The accuracy of the calculation can also be checked by comparing **J** with that generated internally. Another reason for this failure may be the use of an inappropriate MITER, for example, MITER = 3 for a problem that does not have a diagonally dominant Jacobian. It may be helpful to try different values for MITER and monitor the successive corrector estimates stored as the Y array in subroutine STODE.

In addition to the error messages just discussed, a warning message is printed if the step size H becomes so small that T + H = T on the computer, where T is the current value of the independent variable. This error is not considered fatal, and so the execution is not terminated nor is a return made to the calling subprogram. No action is required by the user. The warning message is printed a maximum number of MXHNIL (default value, 10) times per problem. The user can change the number of times the message is printed by resetting MXHNIL, as discussed in section 4.6. To indicate the change to LSODE, the parameter IOPT must be set equal to 1 before LSODE is called.

Chapter 5
Example Problem

5.1 Description of Problem

In this chapter we demonstrate the use of the code by means of a simple stiff problem taken from chemical kinetics. The test case, described elsewhere (refs. 17, 28, and 38), consists of three chemical species participating in three irreversible chemical reactions at constant density and constant temperature:

$$\mathcal{S}_1 \overset{k_1}{\rightarrow} \mathcal{S}_2, \tag{5.1}$$

$$\mathcal{S}_2 + \mathcal{S}_3 \overset{k_2}{\rightarrow} \mathcal{S}_1 + \mathcal{S}_3, \tag{5.2}$$

$$\mathcal{S}_2 + \mathcal{S}_2 \overset{k_3}{\rightarrow} \mathcal{S}_3 + \mathcal{S}_3, \tag{5.3}$$

with $k_1 = 4\times10^{-2}$, $k_2 = 10^4$, and $k_3 = 1.5\times10^7$. In reactions (5.1) to (5.3), $\mathcal{S}_i$ is the chemical symbol for the *i*th species, the arrows denote the directions of the reactions (the single arrow for each reaction means that it takes place in the indicated direction only), and the $\{k_j\}$ are the specific rate coefficients for the reactions. The units of k_j depend on reaction type (e.g., ref. 39). If y_i denotes the molar concentration of species i, that is, moles of species i per unit volume of mixture, the governing ODE's are given by

$$\frac{dy_1}{dt} = -0.04\,y_1 + 10^4 y_2 y_3, \tag{5.4}$$

$$\frac{dy_2}{dt} = 0.04\,y_1 - 10^4 y_2 y_3 - 3\times10^7 y_2 y_2, \tag{5.5}$$

$$\frac{dy_3}{dt} = 3\times10^7 y_2 y_2, \tag{5.6}$$

where t is time in seconds. The initial conditions are

$$y_1(t=0)=1; \quad y_2(t=0)=y_3(t=0)=0. \tag{5.7}$$

The example problem is interesting because the reaction rate coefficients vary over nine orders of magnitude. Also it can be quite easily verified that at steady state, that is, as $t \to \infty$, $y_1 \to 0$, $y_2 \to 0$, and $y_3 \to 1$. To study the evolution of the chemical system, including the approach to the final state, we integrate the ODE's up to $t = 4\times10^{10}$ s, generating output at $t = 0.4\times10^n$ s ($n = 0,1,...,11$).

5.2 Coding Required To Use LSODE

5.2.1 General

All of the coding required to solve the example problem with LSODE is included (in the form of comment statements) in the package supplied to the user. The MAIN program that calls LSODE and manages output is given in figure 5.1. Figure 5.2 lists the subroutine that computes the derivatives. Because a value of MITER = 1 is used (fig. 5.1), a routine that computes the analytical Jacobian matrix is required. This routine is given in figure 5.3. The names used for the derivative and Jacobian matrix subroutines are, respectively, FEX and JEX. Therefore these names are used as arguments in the call to LSODE and declared EXTERNAL in the MAIN program (fig. 5.1).

5.2.2 Selection of Parameters

Because the problem is stiff, the choice METH = 2 is made. For the same reason functional iteration, that is, MITER = 0, is rejected. It is straightforward to compute the analytical Jacobian matrix, which should be used for reasons of efficiency. In any case, the choice MITER = 3, that is, Jacobi-Newton iteration, must not be made because the Jacobian matrix is not diagonally dominant. The choice MITER = 4 with ML = 1 and MU = 2 could be made but will require more storage than MITER = 1 (see table 4.9). More importantly the computational overhead for the LU-decomposition of the iteration matrix is more for MITER = 4 than for MITER = 1. Hence the value MF = 21 is used.

The number NEQ of ODE's is equal to the number (= 3) of chemical species. To minimize storage, the lengths LRW and LIW of the work arrays RWORK and IWORK are set equal to their minimum required values. According to the formulas given in tables 4.9 and 4.10 for MF = 21, these lengths are as follows:

```
      EXTERNAL FEX, JEX
      DOUBLE PRECISION ATOL, RWORK, RTOL, T, TOUT, Y
      DIMENSION Y(3), ATOL(3), RWORK(58), IWORK(23)
      NEQ = 3
      Y(1) = 1.D0
      Y(2) = 0.D0
      Y(3) = 0.D0
      T = 0.D0
      TOUT = .4D0
      ITOL = 2
      RTOL = 1.D-4
      ATOL(1) = 1.D-6
      ATOL(2) = 1.D-10
      ATOL(3) = 1.D-6
      ITASK = 1
      ISTATE = 1
      IOPT = 0
      LRW = 58
      LIW = 23
      MF = 21
      DO 40 IOUT = 1,12
        CALL LSODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
     1     IOPT,RWORK,LRW,IWORK,LIW,JEX,MF)
        WRITE(3,20)T,Y(1),Y(2),Y(3)
 20     FORMAT(7H AT T =,E12.4,6H   Y =,3E15.7)
        IF (ISTATE .LT. 0) GO TO 80
 40     TOUT = TOUT*10.D0
      WRITE(3,60)IWORK(11),IWORK(12),IWORK(13)
 60   FORMAT(/12H NO. STEPS =,I4,11H  NO. F-S =,I4,11H  NO. J-S =,I4)
      STOP
 80   WRITE(3,90)ISTATE
 90   FORMAT(///22H ERROR HALT.. ISTATE =,I3)
      STOP
      END
```

Figure 5.1.—Listing of MAIN program for example problem.

```
      SUBROUTINE FEX (NEQ, T, Y, YDOT)
      DOUBLE PRECISION T, Y, YDOT
      DIMENSION Y(3), YDOT(3)
      YDOT(1) = -.04D0*Y(1) + 1.D4*Y(2)*Y(3)
      YDOT(3) = 3.D7*Y(2)*Y(2)
      YDOT(2) = -YDOT(1) - YDOT(3)
      RETURN
      END
```

Figure 5.2.—Listing of subroutine (FEX) that computes derivatives for example problem.

$$LRW = 22 + 3(5 + 1) + 3(3) + 3^2 = 58$$

and

$$LIW = 20 + 3 = 23.$$

Selection of the error tolerances requires some explanation. A scalar RTOL is used because the same number of significant figures is acceptable for all components. However, because y_2 is expected to be much smaller than both y_1

```
      SUBROUTINE JEX (NEQ, T, Y, ML, MU, PD, NRPD)
      DOUBLE PRECISION PD, T, Y
      DIMENSION Y(3), PD(NRPD,3)
      PD(1,1) = -.04D0
      PD(1,2) = 1.D4*Y(3)
      PD(1,3) = 1.D4*Y(2)
      PD(2,1) = .04D0
      PD(2,3) = -PD(1,3)
      PD(3,2) = 6.D7*Y(2)
      PD(2,2) = -PD(1,2) - PD(3,2)
      RETURN
      END
```

Figure 5.3.—Listing of subroutine (JEX) that computes analytical Jacobian matrix for example problem.

and y_3, an array ATOL, with ATOL(2) much smaller than both ATOL(1) and ATOL(3), is used. For these choices of the RTOL and ATOL types, table 4.2 gives ITOL = 2. Pure relative error control cannot be used because the initial values of both y_2 and y_3 are zero and, as $t \to \infty$, $y_1 \to 0$ and $y_2 \to 0$. Pure absolute error control should not be used because of the widely varying orders of magnitude of the $\{y_i\}$. Note that because a scalar RTOL is used, the MAIN program does not require a DIMENSION statement for this variable.

The remainder of the program calling LSODE is straightforward and self-explanatory. Because the output value for ISTATE is equal to 2 for a normal return from LSODE and no parameter (except TOUT) is reset between calls to LSODE, ISTATE does not have to be reset.

5.3 Computed Results

The output from the program, obtained on the Lawrence Livermore Laboratory's CDC–7600 computer using single-precision arithmetic, is given in figure 5.4. In addition to the results at the specified times, values for the following parameters, which give a measure of the computational work required to solve the problem, are printed at the end: total number of integration steps (STEPS), total number of derivative evaluations (F–S), and total number of Jacobian matrix evaluations and LU-decompositions of the iteration matrix (J–S).

```
AT T =  4.0000E-01   Y =  9.851726E-01  3.386406E-05  1.479357E-02
AT T =  4.0000E+00   Y =  9.055142E-01  2.240418E-05  9.446344E-02
AT T =  4.0000E+01   Y =  7.158050E-01  9.184616E-06  2.841858E-01
AT T =  4.0000E+02   Y =  4.504846E-01  3.222434E-06  5.495122E-01
AT T =  4.0000E+03   Y =  1.831701E-01  8.940379E-07  8.168290E-01
AT T =  4.0000E+04   Y =  3.897016E-02  1.621193E-07  9.610297E-01
AT T =  4.0000E+05   Y =  4.935213E-03  1.983756E-08  9.950648E-01
AT T =  4.0000E+06   Y =  5.159269E-04  2.064759E-09  9.994841E-01
AT T =  4.0000E+07   Y =  5.306413E-05  2.122677E-10  9.999469E-01
AT T =  4.0000E+08   Y =  5.494529E-06  2.197824E-11  9.999945E-01
AT T =  4.0000E+09   Y =  5.129458E-07  2.051784E-12  9.999995E-01
AT T =  4.0000E+10   Y = -7.170592E-08 -2.868236E-13  1.000000E+00

NO. STEPS = 330  NO. F-S = 405  NO. J-S =  69
```

Figure 5.4.—Output from program for example problem.

Chapter 6
Code Availability

The present version of LSODE, dated March 30, 1987, is available in single or double precision. The code has been successfully executed on the following computer systems: Lawrence Livermore Laboratory's CDC–7600, Cray-1, and Cray-X/MP; NASA Lewis Research Center's IBM 370/3033 using the TSS operating sytem (OS), Amdahl 5870 using the VM/CMS OS and the UTS OS, Cray-X/MP/2/4 using the COS and UNICOS operating sytems and the CFT and CFT77 compilers, Cray-Y/MP/8/6128 using UNICOS 6.0 and CFT77, Alliant FX/S, Convex C220 minicomputer using the Convex 8.0 OS, and VAX 11/750, 11/780, 11/785, 6320, 6520, 8650, 8800, and 9410; NASA Ames Research Center's Cray-2 and Cray-Y/MP using the UNICOS operating system and the CFT77 compiler; the Sun SPARCstation 1 using the Sun 4.0 OS; the IBM RISC System/6000 using the AIX 3.1 OS and the XLF and F77 compilers; several IRIS workstations using the IRIX 4.0.1 OS and F77 compiler; and various personal computers under various systems.

The LSODE package is one of five solvers included in the ODEPACK collection of software for ordinary differential equations (ref. 2). The official distribution center for ODEPACK is the Energy Science and Technology Software Center at Oak Ridge, Tennessee. (ESTSC supersedes NESC, the National Energy Software Center at Argonne National Laboratory, in this activity.) Both single- and double-precision versions of the collection are available. Additional details regarding code availability and procurement can be obtained from

Energy Science and Technology Software Center
P.O. Box 1020
Oak Ridge, TN 37831–1020
Telephone: (615) 576–2606

The ODEPACK solvers can also be obtained through electronic mail by accessing the NETLIB collection of mathematical software (ref. 40). Both single- and double-precision versions of ODEPACK are contained in NETLIB. Detailed instructions on how to access and use NETLIB are given by Dongarra and Grosse (ref. 40).

References

1. Hindmarsh, A.C.: LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers. ACM SIGNUM Newsletter, vol. 15, no. 4, 1980, pp. 10–11.
2. Hindmarsh, A.C.: ODEPACK: A Systematized Collection of ODE Solvers. Scientific Computing, R.S. Stepleman, et al., eds., North Holland Publishing Co., New York, 1983, pp. 55–64.
3. Shampine, L.F.; and Gordon, M.K.: Computer Solution of Ordinary Differential Equations: The Initial Value Problem. W.H. Freeman and Co., San Francisco, CA, 1975.
4. Lambert, J.D.: Computational Methods in Ordinary Differential Equations. Wiley & Sons, New York, 1973.
5. Forsythe, G.E.; and Moler, C.B.: Computer Solution of Linear Algebraic Systems. Prentice-Hall, Englewood Cliffs, NJ, 1967.
6. Shampine, L.F.: What is Stiffness? Stiff Computation, R.C. Aiken, ed., Oxford University Press, New York, 1985, pp. 1–16.
7. Shampine, L.F.; and Gear, C.W.: A User's View of Solving Stiff Ordinary Differential Equations. SIAM Rev., vol. 21, 1979, pp. 1–17.
8. Radhakrishnan, K.: A Comparison of the Efficiency of Numerical Methods for Integrating Chemical Kinetic Rate Equations. Computational Methods, K.L. Strange, ed., Chemical Propulsion Information Agency Publication 401, Johns Hopkins Applied Physics Laboratory, Laurel, MD, 1984, pp. 69–82. (Also NASA TM–83590, 1984.)
9. Radhakrishnan, K.: New Integration Techniques for Chemical Kinetic Rate Equations. I. Efficiency Comparison. Combust. Sci. Technol., vol. 46, 1986, pp. 59–81.
10. Gear, C.W.: Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, Englewood Cliffs, NJ, 1971.
11. Shampine, L.F.: Stiffness and the Automatic Selection of ODE Codes. J. Comput. Phys., vol. 54, 1984, pp. 74–86.
12. May, R.; and Noye, J.: The Numerical Solution of Ordinary Differential Equations: Initial Value Problems. Computational Techniques for Differential Equations, J. Noye, ed., North-Holland, New York, 1984, pp. 1–94.
13. Forsythe, G.E.; Malcolm, M.A.; and Moler, C.B.: Computer Methods for Mathematical Computations, Prentice-Hall, Englewood Cliffs, NJ, 1977.
14. Hull, T.E., et al.: Comparing Numerical Methods for Ordinary Differential Equations. SIAM J. Numer. Anal., vol. 9, no. 4, 1972, pp. 603–637.
15. Hindmarsh, A.C.: GEAR: Ordinary Differential Equation System Solver. Report UCID–30001, Rev. 3, Lawrence Livermore Laboratory, Livermore, CA, 1974.
16. Gear, C.W.: Algorithm 407, DIFSUB for Solution of Ordinary Differential Equations. Comm. ACM, vol. 14, no. 3, 1971, pp. 185–190.
17. Byrne, G.D.; and Hindmarsh, A.C.: Stiff ODE Solvers: A Review of Current and Coming Attractions. J. Comput. Phys., vol. 70, no. 1, 1987, pp. 1–62.
18. Gear, C.W.: The Numerical Integration of Ordinary Differential Equations. Math. Comput., vol. 21, 1967, pp. 146–156.
19. Gear, C.W.: The Automatic Integration of Stiff Ordinary Differential Equations. Information Processing, A.J.H. Morrell, ed., North-Holland Publishing Co., New York, 1969, pp. 187–193.
20. Gear, C.W.: The Automatic Integration of Ordinary Differential Equations. Comm. ACM, vol. 14, no. 3, 1971, pp. 176–179.
21. Hindmarsh, A.C.: Linear Multistep Methods for Ordinary Differential Equations: Method Formulations, Stability, and the Methods of Nordsieck and Gear. Report UCRL-51186, Rev. 1, Lawrence Livermore Laboratory, Livermore, CA., 1972.
22. Hindmarsh, A.C.: Construction of Mathematical Software. Part III: The Control of Error in the GEAR Package for Ordinary Differential Equations. Report UCID–30050, Pt. 3, Lawrence Livermore Laboratory, Livermore, CA, 1972.

References

23. Byrne, G.D.; and Hindmarsh, A.C.: A Polyalgorithm for the Numencal Solution of Ordınary Differential Equations ACM Trans. Math. Software, vol. 1, no. 1, 1975, pp. 71–96.
24. Brown, P.N.; Byrne, G.D.; and Hindmarsh, A.C: VODE, A Variable–Coefficient ODE Solver. SIAM J. Sci. Stat. Comput., vol. 10, 1989, pp. 1038–1051.
25. Shampıne, L.F.: Implementation of Implicit Formulas for the Solution of ODEs. SIAM J. Sci Stat. Comput., vol. 1, no. 1, 1980, pp. 103–118.
26. Hall, G.; and Watt, J.M , eds.: Modem Numerical Methods for Ordinary Differentıal Equations. Clarendon Press, Oxford, U.K., 1976.
27. Fınlayson, B.A.: Nonlinear Analysis in Chemical Engineering. McGraw Hill, New York, 1980.
28. Lapidus, L.; and Seinfeld, J.H.: Numerical Solution of Ordinary Differential Equations. Academic Press, New York, 1971
29. Henrici, P.: Discrete Variable Methods in Ordinary Differentıal Equations. Wiley, New York, 1962.
30. Shampine, L.F.: Type-Insensitive ODE Codes Based on Implicit A-Stable Formulas. Math. Comput., vol. 36, no. 154, 1981, pp. 499–510.
31. Ortega, J.; and Rheinbolt, W.C : Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York, 1970
32. Shampine, L.F.: Type-Insensitıve ODE Codes Based on Implicıt A(α)-Stable Formulas Math. Comput., vol. 39, no. 159, 1982, pp. 109-123.
33. Nordsieck, A.: On Numerical Integration of Ordinary Differential Equations. Math. Comput., vol 16, 1962, pp. 22–49.
34. Dongarra, J.J , et al.: LINPACK User's Guide. SIAM, Philadelphia, 1979
35. Jones, R.E.: SLATEC Common Mathematical Library Error Handling Package. Report SAND78–1189, Sandia Natıonal Laboratories, Albuquerque, NM, 1978.
36. Strang, G.: Linear Algebra and Its Applications. Second ed., Academic Press, New York, 1980.
37 Radhakrishnan, K.: LSENS—A General Chemical Kinetics and Sensıtivity Analysis Code for Homogeneous Gas-Phase Reactions. I Theory and Numerical Solution Procedures NASA RP–1328, 1994.
38. Robertson, H.H.: The Solution of a Set of Reaction Rate Equations Numerical Analysıs, An Introduction, J. Walsh, ed., Academic Press, New York, 1966, pp. 178–182.
39. Benson, S.W.: The Foundations of Chemical Kinetics. Robert E Krieger Publishing Co., Malabar, FL, 1982.
40. Dongarra, J J.; and Grosse, E.: Dıstributıon of Mathematıcal Software vıa Electronic Mail Comm. ACM, vol. 30, no 5, 1987, pp. 403–407.

☆U S GOVERNMENT PRINTING OFFICE 1995-650-012-00554

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Reference Publication

4 TITLE AND SUBTITLE Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations	5. FUNDING NUMBERS WU-505-62-52
6. AUTHOR(S) Krishnan Radhakrishnan and Alan C. Hindmarsh	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	8. PERFORMING ORGANIZATION REPORT NUMBER E-5843
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA RP-1327 UCRL-ID-113855

11. SUPPLEMENTARY NOTES
Krishnan Radhakrishnan, Sverdrup Technology, Inc., Lewis Research Center Group, 2001 Aerospace Parkway, Brook Park, Ohio 44142 (work funded by NASA Contract NAS3-25266), and Alan C Hindmarsh, Lawrence Livermore National Laboratory, Livermore, CA 94551 (work funded by DOE Contract W-7405-ENG-48) Responsible person, Edward J Mularz, organization code 2650, (216) 433-5850. Reprinted in October 1994 incorporating corrections noted on the errata

12a DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Categories 61 and 64	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

LSODE, the Livermore Solver for Ordinary Differential Equations, is a package of FORTRAN subroutines designed for the numerical solution of the initial value problem for a system of ordinary differential equations It is particularly well suited for "stiff" differential systems, for which the backward differentiation formula method of orders 1 to 5 is provided. The code includes the Adams-Moulton method of orders 1 to 12; so it can be used for nonstiff problems as well. In addition, the user can easily switch methods to increase computational efficiency for problems that change character For both methods a variety of corrector iteration techniques is included in the code. Also, to minimize computational work, both the step size and method order are varied dynamically This report presents complete descriptions of the code and integration methods, including their implementation It also provides a detailed guide to the use of the code, as well as an illustrative example problem.

14 SUBJECT TERMS First-order ordinary differential equations; Stiff ODE's, Linear multistep methods, Adams-Moulton method, Backward differentiation formula method; Simple iteration, Newton-Raphson iteration, Numerical Jacobians; Accuracy; Error control; Method order selection; Step size selection	15. NUMBER OF PAGES 122 16. PRICE CODE A06

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT