SOFTWARE FOR THE SOLUTION OF STIFF ODE's
BASED ON SECOND DERIVATIVE FORMULAS

by

R. Sacks-Davis

Technical Report No. 145
March 1979
A B S T R A C T

A code based on Enright's second derivative formulas is given for the numerical solution of stiff ODE's. A predictor-corrector approach is used and a fixed-leading coefficient technique is used to implement the variable stepsize formulas. This strategy is efficient for stiff systems and the underlying algorithm is stable and convergent. The code has been designed consistently with the well-known Adams' code STEP (Shampine and Gordon [1975]) and advantage is taken of the similarity between Adams' formulas and Enright's formulas for an efficient implementation.
1. INTRODUCTION

Several methods have been proposed in the past few years for the numerical solution of the initial-value problem:

\[ y'(x) = f(y(x)), \quad y(x_0) = y_0, \]

in stiff ordinary differential equations. A popular approach has been to approximate the differential equation using backward differentiation formulas (Henrici [1962, p. 206]) and examples of codes using these formulas are DIFSUB (Gear [1971]), GEAR (Hindmarsh [1974]), and EPISODE (Byrne and Hindmarsh [1975]). Unfortunately, the order of the formulas used in these methods must be restricted to be at most six. Moreover, the higher-order formulas are not absolutely stable near the imaginary axis. Consequently, methods based on these formulas become relatively less efficient at stringent tolerances or when solving stiff problems that contain eigenvalues with large imaginary parts.

Largely to overcome these difficulties, Enright [1972] proposed a class of second-derivative formulas for the solution of stiff systems. These formulas are more accurate than the corresponding backward differentiation formulas and stiffly stable formulas of orders up to nine exist. However, in order to apply these formulas, an approximation to the second-derivative must be computed every step. Since for autonomous systems \( y'' = \frac{\partial f}{\partial y}. \) current codes based upon these formulas require that the user supply a routine for evaluating the Jacobian as well as one for evaluating the function \( f. \) Note that for stiff systems, some approximation to the Jacobian is required anyway for the solution of the resulting systems of implicit equations, so the requirement of an analytic Jacobian is not a great disadvantage.
In his paper, Enright [1972] presented a code, SDBASIC for the solution of stiff systems. This code has performed quite well in tests that have been carried out (Enright, Hull and Lindberg, [1975]) but because the code uses a one-step-two-half-step error estimate and restricts stepsize changes to halving and doubling, it is apparent that using a predictor corrector approach and estimating the error as a difference between predicted and corrected values would be more efficient.

This is the strategy adopted in the program presented in this paper. A fixed leading coefficient technique is used to implement the variable stepsize formulas. This technique is fully discussed by Jackson and Sacks-Davis [1978] and is efficient for solving stiff systems. Moreover, theoretical results for second derivative formulas indicate that our implementation is stable and convergent (Jackson [1978]). The only restrictions required on the stepsize and order strategies are that both the order of the formulas and the stepsize ratios, \( h_{n+1}/h_n \), be bounded above.

The main routine SDSTEP (Second Derivative STEP) presented in this paper advances the integration one step. The structure of this routine is very similar to the routine STEP (Shampine and Gordon [1975]), which was designed for the solution of non-stiff systems and advances the integration one step using an Adams PECE method. We feel that by designing SDSTEP consistently with such a well known code as STEP we have facilitated easy understanding of our code.

Aside from design considerations there are other similarities between our code and STEP. As in STEP the polynomials used in our method are represented in the code using modified divided differences. Moreover, SDSTEP uses an Adams-Bashforth formula to predict the solution. Thus,
we are able to include in our code many of the techniques developed for STEP for the efficient calculation of the predicted values and other coefficients.

There are other advantages to using an Adams-Bashforth predictor rather than the usual explicit second derivative formula. The structure of our code leads quite naturally to the possibility of designing a code which predicts using an Adams-Bashforth formula, and chooses between an Adam-Moulton corrector or a second derivative corrector depending on whether the equations to be solved are non-stiff or stiff respectively.

In the next section the basic algorithm for advancing a step is described in terms of the polynomials used for predicting and correcting. The actual representation of these polynomials is discussed in Section 3.

In Sections 4 and 5 we discuss the strategies used for predicting the solution and solving the corrector equation. Several possibilities for estimating the local error are presented in Section 6. As well as showing that each of these estimates can be presented as a difference between predicted and corrected values we look at these estimates from the point of view of solving for a higher order approximation to the solution.

In Sections 7 to 11 the detailed strategies used in SDSTEP for choosing the stepsize, choosing the order, starting, updating the divided differences, and testing for possible error returns are described. Many of the strategies used are similar to those in STEP but there are important differences. For example, local extrapolation is not used, the corrector equation is solved to completion, the devices used in STEP for the control of rounding error are not used in SDSTEP and so on. Most of these differences are a consequence of the fact that SDSTEP is designed for stiff systems rather than non-stiff problems.
The code is divided into seven logical units or blocks. A block flowchart for SDSTEP is given in Section 12 together with a description of the contents of each block. In Section 13 we briefly discuss how to implement the code on various computing systems. It should be pointed out that Shampine and Gordon [1975] give a driver routine DE and an interpolation routine INTRP to be used in conjunction with their routine STEP. The corresponding auxiliary routines for SDSTEP are not included in this report.

Finally in Section 14 we present the numerical results obtained after testing SDSTEP on a well known class of 30 stiff problems. We also discuss some possibilities for the future development of better software based on second derivative formulas. A listing of SDSTEP is given in the appendix together with an example of how to use the code to solve a typical problem.
2. POLYNÔMIAL BASIS

In this section we describe the basic algorithm for advancing a step in terms of the polynomials used for prediction and correction.

Suppose that $n$ steps of the integration have been performed and that we have approximations $y_{n+1-i}$ to the true solution $y(x_{n+1-i})$ and approximations $y'_{n+1-i} = f(y_{n+1-i})$ to the slope $y'(x_{n+1-i})$ at the previous points $x_{n+1-i}$, $i = 1, 2, \ldots, n$. It is required to advance a step from $x_n$ to $x_{n+1}$ with stepsize $h_{n+1} = x_{n+1} - x_n$. The first stage of the algorithm is to calculate the predicted value $p_{n+1}$ at $x_{n+1}$. The prediction stage is based on the Adams-Bashforth polynomial of order $k$:

$$
(2.1) \quad p_{n+1} = y_n + \int_{x_n}^{x_{n+1}} p_{k,n}(t) dt
$$

where $p_{k,n}(x)$ is the polynomial of degree $k-1$ satisfying

$$
(2.2) \quad p_{k,n}(x_{n+1-i}) = y'_{n+1-i}, \quad i = 1, 2, \ldots, k.
$$

For the time being, let us assume that $k > 2$. The case of $k = 2$ is discussed separately. In our fixed leading coefficient implementation of the second derivative formulas, the corrector formula may be written in the form

$$
(2.3) \quad y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} c_{k,n}(t) dt
$$

where $c_{k,n}(x)$ is the polynomial of degree $k-1$ satisfying

$$
(2.4) \quad \begin{align*}
 c_{k,n}(x_{n+1}) &= y'_{n+1} \\
 c'_{k,n}(x_{n+1}) &= y''_{n+1} \\
 c_{k,n}(x_{n+1-1h_{n+1}}) &= p_{k,n}(x_{n+1-1h_{n+1}}) \\
 c'_{k,n}(x_{n+1-1h_{n+1}}) &= p'_{k,n}(x_{n+1-1h_{n+1}}) \\
 c''_{k,n}(x_{n+1-1h_{n+1}}) &= p''_{k,n}(x_{n+1-1h_{n+1}}) \\
 &\quad i = 1, 2, \ldots, k-2.
\end{align*}
$$
Here

\begin{equation}
\frac{y''_{n+1}}{y_{n+1}} = \frac{\partial f}{\partial y} \left|_{P_{n+1}} \right. \frac{y'_{n+1}}{y_n}
\end{equation}

is an approximation to the second derivative at \( x_{n+1} \). Of course for autonomous systems, \( y''(x_{n+1}) = \frac{\partial f}{\partial y} \left|_{y(x_{n+1}), y'(x_{n+1})} \right. \). It should be noted that in our fixed leading coefficient implementation

1. only one Jacobian evaluation is required at each attempted step,
2. the corrector polynomial \( C_{k,n}(x) \) interpolates the predictor polynomial at equally spaced past points. Contrast this with a variable coefficient implementation (see, for example Sacks-Davis [1977a]) where the corrector polynomial would be defined by the \( k \) conditions

\begin{align}
C_{k,n}^i(x_{n+1}) &= y_{n+1}^i \\
C_{k,n}^{i+1}(x_{n+1}) &= y_{n+1}^{i+1} \\
C_{k,n}^{i+1}(x_{n+1}-1) &= y_{n+1-i} \\
& \quad i = 1, 2, \ldots, k-2.
\end{align}

3. local extrapolation (for which the corrector polynomial interpolates one extra past value) would not increase the order of accuracy of the method since the corrector polynomial \( C_{k,n}(x) \) is defined in terms of \( P_{k,n}(x) \), a polynomial of degree \( k-1 \).

Using the generalized Lagrangian form of \( C_{k,n}(x) \), equation (2.3) may be written in the form

\begin{equation}
y_{n+1} = y_n + \beta_0 h_{n+1} y_{n+1} + \gamma_0 h_{n+1}^2 y_{n+1} + \sum_{i=1}^{k-2} \beta_i h_{n+1}^{i+1} P_{k,n}(x_{n+1} - ih_{n+1}).
\end{equation}

Here the coefficients \( \beta_i, i = 0, 1, \ldots, k-2 \) and \( \gamma_0 \), are simply the constant coefficients of Enright’s fixed coefficient second derivative
formulas. In our implementation, only the leading coefficients $\beta_0$ and $\gamma_0$ are required by the code. These coefficients are listed in Table 2.1. They are taken from Enright [1974, p.327] where all the coefficients (for $k > 2$) are given.

<table>
<thead>
<tr>
<th>$k$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_0$</td>
<td>1</td>
<td>$\frac{2}{3}$</td>
<td>$\frac{29}{48}$</td>
<td>$\frac{307}{540}$</td>
<td>$\frac{3133}{5760}$</td>
<td>$\frac{317731}{604800}$</td>
<td>$\frac{247021}{483840}$</td>
<td>$\frac{1758023}{3528900}$</td>
</tr>
<tr>
<td>$\gamma_0$</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{6}$</td>
<td>$\frac{1}{8}$</td>
<td>$\frac{19}{180}$</td>
<td>$\frac{3}{32}$</td>
<td>$\frac{863}{10080}$</td>
<td>$\frac{275}{3456}$</td>
<td>$\frac{33953}{453600}$</td>
</tr>
</tbody>
</table>

Table 2.1 The leading coefficients, $\beta_0$ and $\gamma_0$, of Enright's second derivative formulas.

Using (2.2) the corrector equation (2.6) may, in turn, be expressed in the form

$$
(2.7) \quad y_{n+1} = y_n + \beta_0 h_{n+1} y_{n+1} + \frac{k}{2} \beta^{*}_{n+1,j} h_{n+1} y_{n+1-j} + \gamma_0 h_{n+1} y_{n+1}.
$$

Note that although the leading coefficients $\beta_0$ and $\gamma_0$ are fixed, the other coefficients $\beta^{*}_{n+1,j}$, $j = 1, 2, \ldots, k$, depend on the stepsizes taken in the previous $k$ steps.

Rather than calculating $y_{n+1}$ from $y_n$ as in (2.3) we calculate it from the predicted value $y_{n+1}^*$. Subtracting (2.1) from (2.3) gives

$$
(2.8) \quad y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} (C_{k,n}(t) - p_{k,n}(t)) dt.
$$

Let

$$
p_n^{i1} = p_{k,n}(x_{n+1}) \quad \text{and} \quad p_n^{ii} = p_{k,n}'(x_{n+1}).
$$
Then using the interpolatory conditions (2.2) and (2.4) it is easily shown that

\[(2.9) \quad y_{n+1} = p_{n+1} + \beta_0 h_{n+1} (y'_{n+1} - p'_{n+1}) + \gamma_0 h_{n+1}^2 (y''_{n+1} - p''_{n+1}).\]

This is the corrector equation which is solved in SDSTEP at each step.

The case \(k=2\) is similar to the previously discussed cases; the only difference is that the past values \(P_{k,n}(x+h_{n+1})\) do not appear in (2.4) and (2.6). Equation (2.6) becomes

\[(2.10) \quad y_{n+1} = y_n + h_{n+1} y'_{n+1} - \frac{1}{2} h_{n+1}^2 y''_{n+1}.\]

This second order formula is not used by Enright in his implementation. In our method the order is allowed to vary between 2 and 9. Note that on the first step only one previous value is available. Thus an extra starting value must be generated to satisfy (2.2). This case is discussed in Section 9.

3. POLYNOMIAL REPRESENTATION

The polynomial \(P_{k,n}(x)\) defined by (2.2) forms the basis of the prediction stage of our algorithm. However, in Section 2 we did not specify how this polynomial is to be represented by the code. Like Shampine and Gordon [1975] and previously Krogh [1976] we represent \(P_{k,n}(x)\) in terms of simple multiples of the divided differences \([y'_n; y'_{n-1}; \ldots; y'_{n+1-i}]\). This representation allows complete variability of stepsize and is most convenient for the development of a variable order code. Unfortunately, a full description of our code in terms of these "modified" divided
differences does require a lot of notation. In Chapter 5 of their book Shampine and Gordon [1975] give a complete description of their Adams method using a notation based on the variable stepsize \( h_i = x_i - x_{i-1} \) and the modified divided differences. Because we use an Adams-Bashforth predictor we are able to adopt their notation to describe our code. All the definitions given below may be found in Shampine and Gordon [1975, Ch. 5]. Let

\[
\begin{align*}
    h_i &= x_i - x_{i-1} \\
    s &= (x-x_n)/h_{n+1} \\
    \psi_i(n+1) &= h_{n+1} + h_n + \ldots + h_{n+2-i}, \quad i \geqslant 1 \\
    \alpha_i(n+1) &= h_{n+1}/\psi_i(n+1), \quad i \geqslant 1 \\
    \beta_i(n+1) &= 1 \\
    \beta_i(n+1) &= \frac{\psi_i(n+1) \psi_2(n+1) \ldots \psi_{i-1}(n+1)}{\psi_i(n) \psi_2(n) \ldots \psi_{i-1}(n)}, \quad i > 1 \\
    \phi_1(n) &= y_n \\
    \phi_i(n) &= \psi_i(n) \psi_2(n) \ldots \psi_{i-1}(n) [y_n; y_{n-1}; \ldots; y_{n-i+1}], \quad i > 1 \\
    \phi_i(n) &= \beta_i(n+1) \phi_i(n), \quad i \geqslant 1
\end{align*}
\]

\[
c_{i,n}(s) = \begin{cases} 
1 & , i = 1 \\
\frac{sh_{n+1}}{\psi_1(n+1)}, & i = 2 \\
\left( \frac{sh_{n+1}}{\psi_1(n+1)} \right) \left( \frac{sh_{n+1}+\psi_1(n)}{\psi_2(n+1)} \right) \ldots \left( \frac{sh_{n+1}+\psi_{i-2}(n)}{\psi_{i-1}(n+1)} \right), & i \geqslant 3
\end{cases}
\]

\[
g_{i,q} = (q-1)! \int_0^1 \int_0^{s_q-1} \ldots \int_0^{s_1} c_{i,n}(s_0) ds_0 ds_1 \ldots ds_{q-1}.
\]
We refer the reader to Shampine and Gordon [1975, Ch.5] for a full description of how these quantities may be calculated efficiently by the code. They show how advantage may be taken of constant stepsize to reduce the amount of work required at each step. After \( k \) steps of constant stepsize, most of the coefficients remain constant.

Only two more definitions are required. The first is given in Shampine and Gordon [1975, pp. 111-112] and the second is a simple extension of one of the definitions in (3.1). Let

\[
\begin{align*}
\sigma_1(n+1) &= 1 \\
\sigma_i(n+1) &= \frac{(i-1)!h_{n+1}^{i-1}}{\psi_1(n+1)\psi_2(n+1)\ldots\psi_{i-1}(n+1)} , & i > 1 \\
\alpha_1(n+1) &= 0 \\
\alpha_i(n+1) &= \alpha_1(n+1) + \alpha_2(n+1) + \ldots + \alpha_{i-1}(n+1) , & i > 1.
\end{align*}
\]  

(3.2)

\[ \text{A. CALCULATING THE PREDICTED VALUES} \]

In this section we describe the prediction stage of our algorithm. The predicted values \( p_{n+1}, p'_{n+1} \) and \( p''_{n+1} \) of Section 2 are based on the polynomial \( p_k,\tilde{\nu}(x) \) of (2.2). Written in divided difference form

\[
p_{k,\tilde{\nu}}(x) = y_{n}^1 + (x-x_{n}^1)[y_{n}^1; y_{n-1}^1] + \ldots + (x-x_{n}^1)(x-x_{n-1}^1)\ldots(x-x_{n-k+2}^1) \cdot [y_{n}^1; y_{n-1}^1; \ldots; y_{n-k+1}^1] \\
\text{so that a typical term is}
\]

\[
(x-x_{n}^1)(x-x_{n-1}^1)\ldots(x-x_{n-2}^1) \cdot [y_{n}^1; y_{n-1}^1; \ldots; y_{n-k+1}^1] 
\]
Using the notation of (3.1) we may write this term in the form (see also Shampine and Gordon [1975, p.78]).

\[
\begin{align*}
(4.1) \left( \frac{sh_{n+1}}{\psi_1(n+1)} \right) \left( \frac{sh_{n+1} + h_n}{\psi_2(n+1)} \right) \cdots \left( \frac{sh_{n+1} + h_n + \cdots + h_n + h_{n+1}}{\psi_{i-1}(n+1)} \right) \\
\quad \psi_{i+1}(n+1) \psi_2(n+1) \psi_{i-1}(n+1) \\
\quad \psi_1(n) \psi_2(n) \psi_{i-1}(n) \\
\quad \phi_i(n) \\
= c_{i,n}(s) \beta_i(n+1) \phi_i(n) \\
= c_{i,n}(s) \phi_i^*(n).
\end{align*}
\]

Thus, using (2.1), the predicted value \( p_{n+1} \) satisfies

\[
(4.2) \quad p_{n+1} = \gamma_n + \int_{x_n}^{x_{n+1}} p_{k,n}(t) \, dt =
\]

\[
= \gamma_n + h_{n+1} \sum_{i=1}^{k} \phi_i^*(n) \int_{0}^{1} c_{i,n}(s) \, ds
\]

\[
= \gamma_n + h_{n+1} \sum_{i=1}^{k} g_i, I_i \phi_i^*(n).
\]

Also, taking \( s=1 \) in (4.1) and using the fact that \( c_{i,n}(1) = 1 \), \( i = 1,2,\ldots,k \) we have

\[
(4.3) \quad p_{n+1} = p_{k,n}(x_{n+1}) = \sum_{i=1}^{k} \phi_i^*(n).
\]

Finally, after differentiating a typical term (4.1) and substituting \( s=1 \) we have:
After calculating the predicted values using equations (4.2), (4.3) and (4.4) it is possible to solve the corrector equation (2.9) and advance the solution one step.

5. SOLVING THE CORRECTOR EQUATION

At each step it is necessary to solve the corrector equation (2.9).

Let

\[ c_{n+1} = p_{n+1} - \frac{\partial}{\partial y} h_{n+1} p_{n+1}^I - \gamma_o h_{n+1}^2 p_{n+1}^{II} \]

Then, (2.9) may be written in the form

\[ (5.1) \quad \gamma_{n+1} - \frac{\partial}{\partial y} h_{n+1} \gamma_{n+1}^I - \gamma_o h_{n+1}^2 \gamma_{n+1}^{II} + c_{n+1} = 0. \]

or, using (2.5):

\[ (5.2) \quad \gamma_{n+1} - \frac{\partial}{\partial y} h_{n+1} f(\gamma_{n+1}) - \gamma_o h_{n+1} f(\gamma_{n+1}) - c_{n+1} = 0 \]

where

\[ J_{n+1} = \frac{\partial f}{\partial y} \bigg|_{p_{n+1}} \]
The implicit equation (5.2) is solved using a modified Newton-Raphson scheme.

\[ \overline{w}^{(m)}_{n+1} (y_{n+1} - y_{n+1}^{(m)}) = -(y_{n+1} - \beta_0 h_{n+1} f(y_{n+1}^{(m)}) - \gamma_0 h_{n+1}^2 j_{n+1} f(y_{n+1}^{(m)}) - c_{n+1}), \]

\( m = 1, 2, \ldots \)

The iteration matrix \( \overline{w}^{(m)}_{n+1} \) is defined as follows.

Let \( \overline{w}_n \) be the iteration matrix used on the last iteration of the \( n \)th step. Often, this matrix is accurate enough to be used on the current step. However, if a re-evaluation of the iteration matrix is required on the current step (say, on the \( \delta \)th iteration), then

\[ \overline{w}^{(m)}_{n+1} = \begin{cases} 1 - \beta_0 h_{n+1} j_{n+1}^2 - \gamma_0 h_{n+1}^2 j_{n+1}^2, & m \geq \delta \\ \overline{w}_n, & \text{otherwise.} \end{cases} \]

\( \overline{w}^{(m)}_{n+1} \) is stored in its LU factored form for efficient solution of (5.3).

The code is organized so that the linear algebra required to calculate the iteration matrix and to solve the linear systems (5.3) is performed in two auxiliary routines WCALC and WSOLVE. In WCALC, the iteration matrix is re-evaluated (when necessary) and decomposed into its LU factors. The necessary forward and back-substitutions required to solve the linear system

\[ \overline{w}^{(m)}_{n+1} \cdot x = b \]

for \( x \) are performed by the routine WSOLVE. The code is organized this way because for large systems it may be more efficient to use a different factorization of the iteration matrix to the LU decomposition or even a
different iteration matrix (see for example Enright [1976] or Skeel and Kong [1977]). If this is indeed the case, then only subroutines WCALC and WSOLVE need be altered; no change to SDSTEP is necessary.

The strategies used for starting the iteration, updating the iteration matrix and terminating the iteration are listed below.

- The starting value, \( y_{n+1}^{(1)} \), is, on most steps, the predicted value \( P_{n+1} \). However, if after the first step, the order drops down to two, then the starting value is based on the previous two \( y \) values, i.e.

\[
y_{n+1}^{(1)} = y_n + \left( h_{n+1}/h_n \right) \left[ \hat{y}_n; y_{n-1} \right]
\]

for \( k = 2 \) and \( n > 1 \).

- The iteration matrix \( W_{n+1}^{(m)} \) is re-evaluated
  
  (a) on the first iteration, if the order or stepsize has changed,
  
  (b) after four iterations, if a test based on the rates of convergence does not anticipate convergence in one more iteration,
  
  (c) if there is no convergence in five iterations,
  
  (d) if the iteration scheme is diverging.

A maximum of one re-evaluation of the iteration matrix is allowed per (attempted) step.

Convergence is achieved if

\[
\| y_{n+1}^{(m+1)} - y_{n+1}^{(m)} \| \leq 4u \| y_{n+1}^{(1)} \|
\]

where \( u \) is the unit round-off error bound, or if

\[
\| y_{n+1}^{(m+1)} - y_{n+1}^{(m)} \| \leq \epsilon \quad \text{and} \quad \| h_{n+1} g_{k+1} \| \| f(y_{n+1}^{(m+1)}) - f(y_{n+1}^{(m)}) \| \leq \epsilon
\]

Technion - Computer Science Department - Technical Report CS0145 - 1979
where \( \varepsilon \) is the user supplied tolerance. If the first test, (5.5) is satisfied, a function evaluation is saved. The idea behind the second test (5.6) is to keep iterating until further iterations do not significantly affect the error estimate (6.12).

6. ESTIMATING THE LOCAL ERROR

We define the truncation error of the \( k \)-th order formula (2.7) to be

\[
T_k(n+1) = Y(x_{n+1}) - Y(x_n) - \beta_o h_{n+1} Y'(x_{n+1}) - \sum_{j=1}^{k} \beta^{*}_{n+1,j} h_{n+1} Y'(x_{n+1-1}) - \gamma_o h_{n+1}^2 Y''(x_{n+1})
\]

where \( Y(x) \) is the exact solution of \( \{1.1\} \). Throughout this section we assume that \( Y(x) \in C^{k+2} \). In order to estimate the truncation error we derive an alternative expression for \( T_k(n+1) \).

**Proposition 6.1.** The truncation error \( T_k(n+1) \) may be expressed in the form

\[
T_k(n+1) = C_{n+1,k} Y^{(k+1)}(\xi)/k!
\]

where

\[
C_{n+1,k} = h_{n+1} \left\{ g_{k+1,1} - \beta_o - \gamma_o \delta_{k+1}(n+1) \psi_1(n+1) \psi_2(n+1) \ldots \psi_k(n+1) \right\}
\]

and for some \( \xi \in [x_n, x_{n+1}] \).

**Proof.** Using (2.3) we write \( T_k(n+1) \) in the form

\[
T_k(n+1) = Y(x_{n+1}) - Y(x_n) - \int_{x_n}^{x_{n+1}} \xi_{k,n}(t) dt.
\]

where \( \xi_{k,n}(x) \) is the polynomial of degree \( k-1 \) satisfying
\[ \xi_{k,n}(x_{n+1}) = \gamma'(x_{n+1}), \]
\[ \xi_{k,n}(x_{n+1}^-) = \gamma(x_{n+1}), \]
\[ \xi_{k,n}(x_{n+1}^- h_{n+i}) = p_{k,n}(x_{n+1}^- h_{n+i}), \quad i = 1,2,\ldots,k-2 \]

and \( p_{k,n}(x) \) is the polynomial of degree \( k+1 \) satisfying,

\[ p_{k,n}(x_{n+1}^- j) = \gamma'(x_{n+1}^- j), \quad j = 1,2,\ldots,k. \]

Obviously

\[ T_k(n+1) = \int_{x_n}^{x_{n+1}} (\gamma'(t) - \xi_{k,n}(t)) \, dt = \int_{x_n}^{x_{n+1}} (\gamma'(t) - p_{k,n}(t)) \, dt - \int_{x_n}^{x_{n+1}} (\xi_{k,n}(t) - p_{k,n}(t)) \, dt. \]

Now,

\[ \gamma'(x) - p_{k,n}(x) = (x-x_n) (x-x_{n+1}^-) \cdots (x-x_{n+k}^-) \gamma^{(k+1)}(\xi)/k! \]

for some \( \xi \in [x_{n+1-k}; x_{n+1}]. \) Substituting \( x = x_{n+1} \) gives the result

\[ \gamma'(x_{n+1}^-) - p_{k,n}(x_{n+1}^-) = -\psi_1(n+1) \psi_2(n+1) \cdots \psi_k'(n+1) \gamma^{(k+1)}(\xi)/k! \]

Also after differentiating (6.2) and substituting \( x = x_{n+1} \), we have

\[ \gamma''(x_{n+1}) - p''_{k,n}(x_{n+1}) = \frac{1}{(x_{n+1}^- x_n) (x_{n+1}^- x_{n+1}^-) \cdots (x_{n+1}^- x_{n+k}^-) \gamma^{(k+1)}(\xi)/k!} = \]

\[ \frac{1}{h_{n+1}^2} \sum_{i=1}^{k} \omega_{k+1}(n+1) \psi_1(n+1) \psi_2(n+1) \cdots \psi_k(n+1) \gamma^{(k+1)}(\xi)/k! \]

With these preliminary results we can simplify the expression (6.1).

Using (3.1) and (6.2), the first integral
(6.5)  \[ \int_{x_n}^{x_{n+1}} (y'(t) - p_k,n(t)) dt = \]
\[ = h_{n+1}^{k} \int_{x_n}^{x_{n+1}} (s) \psi_1(n+1) \psi_2(n+1) ... \psi_{k}(n+1) y^{(k+1)}(\xi)/k! ds = \]
\[ = h_{n+1}^{k} o_{k+1} n \psi_1(n+1) \psi_2(n+1) ... \psi_{k}(n+1) y^{(k+1)}(\xi)/k! \]

Obviously,

(6.6)  \[ \int_{x_n}^{x_{n+1}} (e_{k,n}(t) - p_k,n(t)) dt = \]
\[ = h_{n+1}^{k} o_{k,n}(y'(x_{n+1}) - p_k,n(x_{n+1})) + h_{n+1}^{2} \psi_1(n+1) y^{(n+1)}(x_{n+1}) - p_k,n(x_{n+1})) \]
\[ = h_{n+1}^{2} o_{k} + h_{n+1}^{k} o_{k+1}(n+1) ] \psi_1(n+1) \psi_2(n+1) ... \psi_{k}(n+1) y^{(k+1)}(\xi)/k! \]

using (6.3) and (6.4). The result follows from (6.1), (6.5) and (6.6).

Using Proposition (6.1), a natural estimate for \( T_k(n+1) \) is

(6.7)  \[ \text{Est}_1 = C_{n+1,k} [y_{n+1}, y_{n+1}^1, ... ; y_{n+1-k+1}^1] \]
\[ = h_{n+1}^{k} [g_{k+1}, 1 - o_{k+1}(n+1)] y_{k+1}^1(n+1). \]

\( \text{Est}_1 \) may be calculated as a difference between predicted and corrected values as follows (see Equation (10.1)).

(6.8)  \[ \text{Est}_1 = h_{n+1}^{k} [g_{k+1}, 1 - o_{k+1}(n+1)] (y_{n+1}^1 - p_{n+1}^1). \]

\( \text{Est}_1 \) can also be expressed as a difference between the corrected solution \( y_{n+1} \) and another approximation, \( u_{n+1} \), to the solution at \( x_{n+1} \). Consider the multistep formula
(6.9) \[ y_{n+1}^+ = y_n + \beta_0^+ h_{n+1} (y_{n+1})^1 + \sum_{j=1}^{k} \beta_j^+ h_{n+1,j} y_{n+1-j} + \gamma_o h_{n+1}^2 (y_{n+1})^{\nu} - c_{n+1,k}[y_{n+1}^1;y_n^1;\ldots;y_{n-k+1}^1]. \]

This formula is of order \( k+1 \) and may be written in the form

(6.10) \[ y_{n+1}^+ = y_n + \beta_{n+1,0}^+ h_{n+1} (y_{n+1})^1 + \sum_{j=1}^{k} \beta_{n+1,j}^+ h_{n+1,j} y_{n+1-j} + \gamma_o h_{n+1}^2 (y_{n+1})^{\nu}. \]

Like Enright's formulas, (6.10) is stable at infinity. The coefficients \( \beta_{n+1,j}, j = 0, \ldots, k \) are uniquely determined by the requirement that the formula be of order \( k+1 \). (In fact the coefficients are easily determined from the fact that (6.10) can be expressed as a linear combination of the variable coefficient, \((k+1)\)-st order Enright and Adams-Moulton formulas.)

Suppose we have solved (2.9) for \( y_{n+1}^+ \) and we wish to determine the solution of the higher order formula (6.9). If we solve for \( y_{n+1}^+ \) using successive iteration, with starting value \( y_{n+1}^+ \), the first iterate, \( u_{n+1} \) satisfies

\[ u_{n+1} = y_n + \beta_0 h_{n+1} y_n^1 + \sum_{j=1}^{k} \beta_j h_{n+1,j} y_{n+1-j} + \gamma_o h_{n+1}^2 y_{n+1}^1 - c_{n+1,k}[y_{n+1}^1;y_n^1;\ldots;y_{n-k+1}^1] = y_{n+1}^+. \]

We have proved:

**Proposition 6.2** The error estimate \( \text{Est}_1 \) may be calculated as a difference between predicted and corrected values:

\[ \text{Est}_1 = h_{n+1}^1 [g_{k+1,1} - \beta_0 - \gamma_o k+1(n+1)](y_{n+1}^1 - \phi_{n+1}^1). \]
As well,

\[ \text{Est}_1 = Y_{n+1} - u_{n+1} \]

where \( u_{n+1} \) is that value obtained by solving for \( Y_{n+1}^+ \) using one step of successive iteration with starting value \( Y_{n+1} \).

Rather than solve for \( Y_{n+1}^+ \) using successive iteration, suppose we solve for \( Y_{n+1}^+ \) using a modified Newton scheme. This is the basis for the error estimate \( \text{Est}_2 \) defined by

\[ \text{Est}_2 = \tilde{W}_{n+1}^{-1} C_{n+1, k} [y_{n+1}^t; y_{n+1}^t; \ldots; y_{n-k+1}^t] = \tilde{W}_{n+1}^{-1} \text{Est}_1. \]

If we solve (6.9) for \( Y_{n+1}^+ \) using a modified Newton scheme with starting value \( Y_{n+1} \) and iteration matrix \( \tilde{W}_{n+1} \) the first iterate, \( v_{n+1} \), satisfies

\[ v_{n+1} = y_{n+1} - \tilde{W}_{n+1}^{-1} \left[ y_{n+1} - y_{n} - \beta \cdot h_{n+1} \right] \sum_{j=1}^{k} \beta_j \cdot h_{n+1}^j \cdot y_{n+1-j}^t + \tilde{W}_{n+1}^{-1} \left[ y_{n+1}^t; y_{n+1}^t; \ldots; y_{n-k+1}^t \right] = y_{n+1} + \text{Est}_2. \]

We have proved.

**Proposition 6.3.**

The error estimate \( \text{Est}_2 \) may be calculated as a difference between predicted and corrected values as follows:

\[ \text{Est}_2 = h_{n+1} \left[ g_{k+1}^t, \beta_o - \gamma_o \cdot \tilde{w}_{n+1}^{-1} \left( Y_{n+1}^t - p_{n+1}^t \right) \right]. \]

Also

\[ \text{Est}_2 = v_{n+1} - Y_{n+1} \]
where \( v_{n+1} \) is the value obtained by solving for \( y_{n+1}^+ \) using one iteration of a modified Newton scheme with starting value \( y_{n+1}^- \) and iteration matrix \( \tilde{W}_{n+1} \).

Note that both \( \text{Est}_1 \) and \( \text{Est}_2 \) are asymptotically equivalent to the local error. Noting that

\[
T_k(n+1) = \int_{X_n}^{X_{n+1}} (\gamma'(t) - \xi_{k,n}(t)) dt
\]

another approach to deriving computable, asymptotically correct error estimates, is to approximate \( \xi_{k,n}(t) \) by \( C_{k,n}(t) \) and to approximate \( \gamma'(t) \) by a polynomial of degree one higher than \( C_{k,n}(t) \). (See, for example Sedgwick [1973].) In our context, a convenient estimate is,

\[
\text{Est}_3 = \int_{x_n}^{x_{n+1}} (P_{k+1,n+1}(t) - C_{k,n}(t)) dt.
\]

where \( P_{k+1,n+1}(x) \) is the polynomial of degree \( k \) satisfying

\[
P_{k+1,n+1}(x) = \gamma'_{n+2}(x), \quad l = 1, 2, \ldots, k+1.
\]

Writing

\[
\text{Est}_3 = \int_{x_n}^{x_{n+1}} (P_{k+1,n+1}(t) - P_{k,n}(t)) dt - \int_{x_n}^{x_{n+1}} (C_{k,n}(t) - P_{k,n}(t)) dt
\]

and noting that

\[
P_{k+1,n+1}(x) - P_{k,n}(x) = \frac{(x-x_n)(x-x_{n-1})\ldots(x-x_{n-k+1})}{(x_{n+1}-x_n)(x_{n+1}-x_{n-1})\ldots(x_{n+1}-x_{n-k+1})} (y_{n+1}^+ - p_{n+1}^-)
\]

we have, using (2.8)
\[ \text{Est}_3 = h_{n+1}g_{k+1,1}(y_{n+1}^1 - p_{n+1}^1) - (y_{n+1} - p_{n+1}). \]

Analogously to \( \text{Est}_2 \) we define the estimate

\[ \text{Est}_4 = \frac{1}{n+1} \text{Est}_3. \]

Unlike the error estimates \( \text{Est}_1 \) and \( \text{Est}_2 \), which may be viewed as iterating towards the formula (6.9), these latter two estimates may be viewed as iterating towards the Adams-Moulton formula

\[ y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} p_{k+1,1}(t) dt \]

or order \( k+1 \). We summarize these results below:

**Proposition 6.4.** The error estimates \( \text{Est}_3 \) and \( \text{Est}_4 \) may be calculated as a difference between predicted and corrected values

\[ \text{Est}_3 = h_{n+1}g_{k+1,1}(y_{n+1}^1 - p_{n+1}^1) - (y_{n+1} - p_{n+1}) \]

and

\[ \text{Est}_4 = \frac{1}{n+1} \left[ h_{n+1}g_{k+1,1}(y_{n+1}^1 - p_{n+1}^1) - (y_{n+1} - p_{n+1}) \right]. \]

Both of these estimates may be expressed in the form \( y_{n+1} - w_{n+1} \) where \( w_{n+1} \) is that value obtained by taking one step of an iterative method towards the Adams-Moulton formula of order \( k+1 \).

In deciding which of the above estimates to include in the code, several factors need to be considered.

(i) The error estimates \( \text{Est}_3 \) and \( \text{Est}_4 \) are more expensive to evaluate. Since \( \tilde{w}_{n+1} \) is stored in \( LU \)-factored form an extra forward and back substitution is required at each step.
Experimental results indicate that Est₁ and Est₃ can overestimate errors in the transient components of a stiff system. On the examples in this paper, it was found that using Est₄ rather than Est₃ lead to a decrease in the number of steps (and function and Jacobian evaluations and LU decompositions) ranging from 0-10%, with almost no difference in the accuracy of the computed results.

The performance of these error estimates is quite dependent on other heuristics used in the program.

Further discussion about similar estimates for stiff systems is given in Sacks–Davis [1977b].

In the program given in the appendix, Est₄ is used. The quantity \( h_{n+1} g_{k+1} (y_{n+1}, \ldots, P_{n+1}) - (y_{n+1}, P_{n+1}) \) is calculated and stored in DD. Then Est₃ is calculated by a call to WSOLVE. In order to change the code so that Est₃ is used rather than Est₄, it is simply required to delete the statement:

```
call WSOLVE (NEQN, WBAR, NPLV, DD)
```

from block 4. Similarly, the changes required to use Est₁ or Est₂ are quite trivial.
Finally, we calculate the weighted norm

\[(6.12)\quad \text{ERR} = \|\text{Est}_q\| =
\|
\begin{pmatrix} h_{n+1} g_{k+1} \\ (y_{n+1} - p_{n+1}) \\ (y_{n+1} - p_{n+1}^*) \end{pmatrix}
\|
\]

7. CHOOSING STEPSIZE

The step choosing strategy in SDSTEP is based on the estimate of the local error \(\text{ERR} = \|\text{Est}_q\|\). Assuming that the error estimate does not overestimate the relatively small errors in the transient components, the usual asymptotic arguments imply that the appropriate stepsize for the next step is \(r h_{n+1}\), where \(r = (\text{ERR}/\varepsilon)^{-1/(k+1)}\) and \(\varepsilon\) is the local error tolerance and \(k\) is the order.

The conservative choice of

\[(7.1)\quad r = 0.95 \times (\text{ERR}/0.5\varepsilon + 0.0001)^{-1/(k+1)}\]

is used in the code. However, a number of heuristics are also involved in choosing the stepsize and they are listed below:

- after a successful step, the stepsize is increased only if \(r \geq 1.5\) and \(r\) is restricted so that \(r \leq \text{RMAX}\) where \(\text{RMAX} = 2.0\).

- after a failure, the cause of the failure is determined. If the cause of the failure was the non-convergence of the Newton scheme (CONVGD = FALSE), the stepsize is reduced by a factor of 4. If the failure was due to the error estimate exceeding the tolerance, then \(r\) is determined as in (7.1) but the factor by which the stepsize is reduced must lie between 9/10 and 1/10.

- in the case of multiple failures, \(r\) is reduced by a factor of 4, after more than three failures, though, this factor becomes 10.

- the strategy for choosing the stepsize on the first step is described in Section 9.
8. CHOOSING ORDER

Consistent with the approach of Shampine and Gordon [1975, Ch. 7] we estimate the local error for different orders assuming constant stepsize. After \( k-2 \) steps of constant stepsize:

\[
P_{k,n}(x_{n+1} - i h_{n+1}) = P_{k,n}(x_{n+1-i}) = y_{n+i-1} \quad i = 1, 2, \ldots, k-2
\]

and the corrector formula (2.6) reduces to Enright's fixed coefficient formula:

\[
y_{n+1} = y_n + \sum_{i=0}^{k-2} \beta_i h_{n+1} y_{n+i-1} + \gamma_0 h_{n+1}^2 y_{n+1}.
\]

Using (8.1), it is easy to show that the truncation error

\[
T_{k,n}(x_{n+1}) = \int_{x_n}^{x_{n+1}} (y'(t) - \xi_{k,n}(t)) dt
\]

where \( \xi_{k,n}(x) \) is the polynomial of degree \( k-1 \) satisfying

\[
\xi_{k,n}(x_{n+i-1}) = y'(x_{n+i-1}), \quad i = 0, 1, \ldots, k-2
\]

and

\[
\xi_{k,n}(x_{n+1}) = y''(x_{n+1}).
\]

It follows that

\[
T_{k,n}(x_{n+1}) = \Pi_{k,n+1}^* (k+1)(\xi)
\]

for some \( \xi \in [x_n, x_{n+3}, x_{n+1}] \) where

\[
\Pi_{k,n}^* = \int_0^1 (s-1)^2 s(s+1) \ldots (s+k-3) ds / k!
\]

are the error constants of Enright's formulas. Application of
proposition (6.1) gives

\[ \pi_k^* = g_{k+1} \beta_0 - \beta_0 g_{k+1} (n+1). \]

Now, when the stepsize is constant \( g_{k+1} \), it reduces to an error constant of the k-th order Adams-Bashforth formula (see for example Shampine and Gordon [1977, p. 83]). Also \( \tilde{a}_{k+1} (n+1) = 1 + \frac{1}{2} + \ldots + \frac{1}{k} \). Hence, the \( \pi_k^* \) are easily calculated. They are listed below to 3 sig. figs. for \( k=2, \ldots, 9 \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^3 \pi_k^* )</td>
<td>167</td>
<td>13.9</td>
<td>4.86</td>
<td>2.36</td>
<td>1.36</td>
<td>0.863</td>
<td>0.590</td>
<td>0.424</td>
</tr>
</tbody>
</table>

Using (8.2) we may estimate \( T_k (n+1) \) under the assumption of constant stepsize by forming the quantity

\[ \text{ERK} = \left\| h_{n+1} \tilde{a}_{k+1} (n+1) \tilde{a}_{k+1} (n+1) \right\| \]

where \( \tilde{a}_{k+1} (n+1) \) is defined by (3.2).

Similarly, we form

\[ \text{ERKM1} = \left\| h_{n+1} \tilde{a}_{k} (n+1) \tilde{a}_{k} (n+1) \right\| \]

and

\[ \text{ERKM2} = \left\| h_{n+1} \tilde{a}_{k-1} (n+1) \tilde{a}_{k-1} (n+1) \right\| \]

for orders \( k-1 \) and \( k-2 \) respectively. The higher-order estimate is formed only after \( k+1 \) steps of constant stepsize, in which case \( \tilde{a}_{k+2} (n+1) = 1 \).
Thus the error estimate for higher order is

$$ERK_{P1} = \| h_{n+1} h_{k+1} \phi_{k+2}(n+1) \|$$

Note that we have not included the factor $W_{n+1}^{-1}$ in these estimates as is done in the formation of $ERR$, the local error estimate. The extra expense is not justified since it is the relative sizes of these estimates which determine the order on the next step. Looked at from another point of view, the aim of our strategy is to ensure that the modified divided differences, $\phi_i(n+1)$, which are used in the prediction stage of the next step do not start increasing in norm with $i$.

The strategies for raising or lowering order are, with two exceptions, identical to those used in STEP and described in Shampine and Gordon [1975, pp. 12-120]. As in STEP there is an initial phase when the order is increased every step and after that the basic strategy is to increase or decrease the order if the error estimates $ERK_{M2}$, $ERK_{M1}$, $ERK$ and $ERK_{P1}$ form a decreasing or increasing sequence respectively. The two exceptions are:

1. The order varies from 2 to 9 rather than from 1 to 12.
2. After the initial phase (PHASE1 = TRUE) the order cannot be increased on two steps in succession.
9. STARTING

Before the first step of the integration can be taken, an appropriate stepsizes must be determined and other parameters must be initialized. The user supplies an upper bound, $h_{\text{input}}$, on the size of the first step. However, the code automatically determines an appropriate initial step-size internally at the cost of one function evaluation.

A Taylor series formula is used to take a trial step of size $h_{\text{temp}} = 2 \sqrt{u}$ where $u$ is the unit round-off error bound. Let

$$y_{\text{trial}} = y_0 + h_{\text{temp}} y'_0 + \frac{1}{2} h_{\text{temp}}^2 \cdot \text{DD}$$

where the divided difference

$$\text{DD} = [y'_0; y'_0] = \frac{3 f}{3 y} |_{y = y'_0}$$

is the approximation to the second derivative at $x_0$. Let

$$y'_0 = f(y_{\text{trial}})$$

The weighted norm $\|y''''(x_0)/6\|$ is estimated by calculating

$$\text{SUM} = \| \frac{1}{3} [y'_0; y'_0; y'_0; y'_0] \|.$$  

Because we estimate that the error made in taking a step of size $h$ with the second order formula (2.9) will be approximately $\|h^3 y''''(x_0)/6\|$, and as we are aiming for an error of $\varepsilon$, the appropriate stepsize for the first step is $(\varepsilon/\text{SUM})^{1/3}$. The code is conservative and chooses

$$h = \min[h_{\text{input}}, \frac{1}{2} (0.5 \varepsilon/\text{SUM})^{1/3}]$$.
Consistent with (2.2), the prediction stage on the first step should be based on the polynomial

\[ P_{2,0}(x) = y_0^i + (x - x_0)^2[y_0^i; y_{-1}^i] \]

and to start the method the values \( \Phi_1^*(1) = y_0^i \) and \( \Phi_1^*(2) = h[y_0^i; y_{-1}^i] \) must be generated. In SDSTEP we initialize \( \Phi_1^*(1) \) to be \( y_0^i \) and \( \Phi_1^*(2) = h[y_0^i; y_{-1}^i] \). Interpreted consistently with (9.1), the extra starting value \( y_{-1}^i \) used in SDSTEP is that approximation to \( y(x_0 - h) \) obtained by evaluating the polynomial \( y_0^i + (x-x_0)^2[y_0^i; y_{-1}^i] \) at \( x_0 - h \).

Rather than setting all the other coefficients as if one step of size \( h \) has been taken it is simpler to set \( H_{\text{PAST}} = 0 \) and allow the code to compute all the coefficients in block 1 automatically.

10. UPDATING THE MODIFIED DIVIDED DIFFERENCES

After each successful step, it is necessary to form the modified divided differences \( \phi_i(n+1) \), \( i = 1, 2, \ldots, k+1 \) from the values \( \phi_i^*(n) \), \( i = 1, 2, \ldots, k \). Using (2.2) and (6.1) it follows that

\[
P_{k+1,n+1}(x) - P_{k,n}(x) = (x-x_n^i)(x-x_{n-1}^i) \ldots (x-x_{n-k+1}^i)[y_{n+1}^i; y_n^i; \ldots; y_{n-k+1}^i].
\]

Substituting \( x = x_{n+1} \) gives

\[
y_{n+1}^i - P_{n+1}^i = \psi_1(n+1) \psi_2(n+1) \ldots \psi_k(n+1)[y_{n+1}^i; y_n^i; \ldots; y_{n-k+1}^i] = \phi_{k+1}^*(n+1).
\]
Thus

$$\phi_{k+1}(n+1) = y_{n+1} - p_{n+1}$$

Also

$$\phi_{1+1}(n+1) = \psi_1(n+1) \psi_2(n+1) \cdots \psi_{1-1}(n+1) [y_{n+1} ; y_n ; \cdots ; y_{n-1+2}] =$$

$$= \psi_1(n+1) \psi_2(n+1) \cdots \psi_{1-1}(n+1) [y_{n+1} ; y_n ; \cdots ; y_{n-1+2}] -$$

$$\frac{\psi_1(n+1) \psi_2(n+1) \cdots \psi_{1-1}(n+1)}{\psi_1(n) \psi_2(n) \cdots \psi_{1-1}(n)} \psi_1(n) \psi_2(n) \cdots \psi_{1-1}(n)$$

$$\cdots [y_n ; y_{n-1} ; \cdots ; y_{n-1+2}] =$$

$$= \phi_1(n+1) - \beta_1(n+1) \phi_1(n)$$

$$= \phi_1(n+1) - \phi_1^*(n).$$

Thus

$$\phi_1(n+1) = \phi_{1+1}(n+1) + \phi_1^*(n).$$

Using equation (10.1) and then equation (10.2) with $i = k, k-1, \ldots, 1$ all the differences can be updated; this approach is very efficient and is very economical of storage.
11. COMPUTER ARITHMETIC

Two tests exist in STEP to prevent the code from attempting to achieve accuracies that are unreasonable because of machine precision. In stepping from $x_n$ to $x_n + h$, the code checks whether

$$ h \geq 4u|x_n| $$

where $u$ is the unit round-off, error bound and whether

$$ 0.5e < 2u|y_n| $$

where $e$ is the user supplied tolerance. These tests are explained in Shampine and Gordon [1975, pp. 146-149] and are incorporated in SDSTEP. If either of these tests fails, control is returned to the calling program with CRASH = .TRUE. and an appropriate stepsize and tolerance for continuing the integration are given.

12. PROGRAM STRUCTURE

The code is divided into seven logical units or blocks. Each block is entered through its first statement and, with the exception of error returns, left through its last statement. A block flowchart for SDSTEP appears in Figure 12.1. A description of the contents of each block appears below:

**Block 0:** The test, described in Section 11, is made to determine whether the stepsize and tolerance are appropriate for continuing the integration given the limitations of machine precision. If the input values are not acceptable a return is made; the input parameters are assigned larger values and the variable CRASH is set to .TRUE. If this is the first step
Figure 12.1 Block flowchart for SDSTEP
of the integration an initial stepsize is calculated and other parameters are set as described in Section 9.

Block 1: This block computes the coefficients of Section 3 as described by Shampine and Gordon [1975, Ch. 5, and pp. 111-112] and is block 1 of their routine STEP.

Block 2: The array PHI is updated to contain $\phi_i(n)$ rather than $\phi_i(n)$. Then the predicted values $p(x) = p_{n+1}$, PD($x$) = $p_{n+1}'$, and HPDD($x$) = $h_{n+1}p_{n+1}''$ are calculated as described in Section 4.

Block 3: The corrector equation (2.9) is solved using the modified Newton scheme described in Section 5. The array ICOEFF($x$) is set in a data statement and holds the values required for calculating the leading coefficients of the second derivative formulas.

The coefficients of the $k$-th order formula are

- $b_0 = ICOEFF(K-1)/ICOEFF(K+7)$ and
- $\gamma_0 = ICOEFF(K+15)/ICOEFF(K+23)$.

The code determines HBEFA = $h_{n+1}b_0$, HGIAMMA = $h_{n+1}\gamma_0$, and HSQGAM = $h_{n+1}^2\gamma_0$.

The counter WCALC is initially set to zero but is set to one if the iteration matrix WBAR is re-evaluated on the current step. The iteration matrix is calculated and factored into LU form in SUBROUTINE WCALC.

In order to solve equation (5.3) for $y_{n+1}^{(m+1)}$, the residual

$-(y_{n+1}^{(m)} - b_0 h_{n+1} f(y_{n+1}^{(m)}) - \gamma_0 h_{n+1}^2 f(y_{n+1}^{(m)}) - c_{n+1})$ is calculated and the forward and back substitutions necessary to solve the linear system (5.3) are then done in SUBROUTINE WSOLVE. The norms YDNORM = $\|y_{n+1}^{(m+1)} - y_{n+1}^{(m)}\|$ and FDNORM = $\|h_{n+1}^k g_{k+1,1}(f(y_{n+1}^{(m+1)}) - f(y_{n+1}^{(m)})\|)$ are calculated so that

Technion - Computer Science Department - Technical Report CS0145 - 1979
the convergence tests (5.5) and (5.6) can be made.

The rate of convergence is estimated by the variable

\[
\text{CRATE} = \frac{\|y_{n+1}^{(m)} - y_{n+1}^{(m-1)}\|}{\|y_{n+1}^{(m)} - y_{n+1}^{(m-1)}\|} \quad \text{on the first iteration, } m=1,
\]

the denominator is set to $10^{10}$. After four iterations we also form

\[
\text{FCRATE} = \frac{\|f(y_{n+1}^{(m+1)}) - f(y_{n+1}^{(m-1)})\|}{\|f(y_{n+1}^{(m)}) - f(y_{n+1}^{(m-1)})\|}.
\]

If the iteration scheme converges (CONVGD = .TRUE.) control goes to Block 4, otherwise it goes to Block 5.

**Block 4:** The estimates of the local errors at orders $k$, $k-1$ and $k-2$, assuming constant stepsize, are calculated. These estimates are defined in Section 8. The modified differences $\phi_{k+1}(n+1)$, $\phi_k(n+1)$ and $\phi_{k-1}(n+1)$ are evaluated using equations (10.1) and (10.2) and are stored in TEMP1, TEMP2, and TEMP3 respectively. The error constants $\tau_k^*$ are stored in the array PISTAR(*) which is set in a data statement at the start of the routine. The error estimate ERR defined by (6.12) is then calculated as described in Section 6.

The possibility of lowering order is considered. A step is accepted if ERR $\leq$ EPS; otherwise, control goes to Block 5.

**Block 5:** The parameters $X$, PSI(*') and PHI(*;*) are restored when a step is unsuccessful. An appropriate stepsize and order for continuing the integration are calculated using the strategies of Sections 7 and 8. The new stepsize is not permitted to be less than four units of round-off error relative to the independent variable (as described in Section 11).
Block 6: After a successful step, a test is made to determine whether the order can be increased. The order and stepsize are then chosen for continuation of the integration. The solution is updated and the divided differences for the next step are calculated using equations (10.1) and (10.2).

13. IMPLEMENTATION

Three routines, `SDSTEP`, `WCALC` and `WSOLVE`, are listed in the appendix. The codes are written in `DOUBLE PRECISION` and have been tested on an IBM 360. In order to change the code to `SINGLE` precision, it is simply required to delete the statement

```
IMPLICIT REAL*:8 (A-H, O-Z)
```

from each of the subroutines and change FORTRAN-supplied functions like DABS to their single-precision equivalents.

The floating-point functions used in `SDSTEP` are DABS, DSQRT, DSIGN, DMAX1 and DMIN1. There are no FORTRAN-supplied functions used by routines `WCALC` and `WSOLVE`.

Subroutines for decomposing a matrix into its LU factors by Gaussian elimination and for performing the forward and back substitutions to solve a linear system must be provided. In testing our routines, we used the subroutines

```
DECOMP(NDIM,N,A,COND,IPVT,WORK)
```

and

```
SOLVE(NDIM,N,A,B,IPVT)
```

listed in Forsythe, Malcolm and Moier [1977]. The routine `DECOMP` returns
an estimate of the condition number of $A$ each time it is called. This involves solving an extra two sets of equations. As the routine DECOMP is called repeatedly by the code, it may be cheaper to use alternate routines for solving the linear systems. This is simple to accomplish. The CALL statements in WCALC and WSOLVE and the test for singularity in WCALC may require modifications.

Two machine-dependent constants, TWOU and FOURU, which represent, respectively, two-times and four-times unit roundoff error $u$ need to be supplied. Appropriate values for many computer systems are listed in Shampine and Gordon [1975, p. 157].

All the FORTRAN conventions used in the codes are consistent with those of Shampine and Gordon [1975, p. 157-158]. An example of how to use SDSTEP to solve a typical problem is given in the appendix.

14. NUMERICAL RESULTS

The following results were obtained after testing SDSTEP on a well known set of test problems: STIFF DETEST (Enright, Hull and Lindberg [1975], Enright and Hull [1976]). There are six classes of problems:

CLASS A - Linear with real eigenvalues
CLASS B - Linear with non-real eigenvalues
CLASS C - Non-linear coupling
CLASS D - Non-linear with real eigenvalues
CLASS E - Non-linear with non-real eigenvalues
CLASS F - Non-linear chemical kinetics problems.
There are 30 problems in all and each problem was solved at four tolerances $10^{-2}$, $10^{-4}$, $10^{-6}$, and $10^{-8}$. The results were obtained on an IBM 360 (University of Toronto) in DOUBLE PRECISION. The summaries for each class is given in Table 14.1. The table headings refer to the time in seconds, number of steps, function calls, Jacobian calls, LU decompositions, the maximum local error, the percentage of steps for which the local error exceeded the tolerance (PERC:DEC.) and for which the local error exceeded 5 times the tolerance (PERC:B:DEC.) for each problem class. Also given are the summaries for each tolerance and a total summary.

<table>
<thead>
<tr>
<th>Class</th>
<th>FCN</th>
<th>JAC.</th>
<th>LU</th>
<th>MAXM.</th>
<th>PERC.</th>
<th>PERC.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Steps</td>
<td>Calls</td>
<td>Calls</td>
<td>Decomp.</td>
<td>LOC. ERR.</td>
</tr>
<tr>
<td>A</td>
<td>6.394</td>
<td>1548</td>
<td>3232</td>
<td>1606</td>
<td>603</td>
<td>7.2</td>
</tr>
<tr>
<td>B</td>
<td>8.162</td>
<td>3443</td>
<td>6994</td>
<td>3495</td>
<td>.772</td>
<td>1.8</td>
</tr>
<tr>
<td>C</td>
<td>4.815</td>
<td>2170</td>
<td>6656</td>
<td>2244</td>
<td>790</td>
<td>2.7</td>
</tr>
<tr>
<td>D</td>
<td>3.445</td>
<td>1596</td>
<td>5957</td>
<td>1759</td>
<td>1181</td>
<td>27.2</td>
</tr>
<tr>
<td>E</td>
<td>4.402</td>
<td>1943</td>
<td>6667</td>
<td>2101</td>
<td>854</td>
<td>7.0</td>
</tr>
<tr>
<td>F</td>
<td>8.241</td>
<td>3383</td>
<td>13621</td>
<td>3719</td>
<td>1999</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Summary: 354.49 | 14083 | 43127 | 14924 | 6199 | 27.2 | 0.7 | 0.0 |

<table>
<thead>
<tr>
<th>Tol</th>
<th>FCN</th>
<th>JAC.</th>
<th>LU</th>
<th>MAXM.</th>
<th>PERC.</th>
<th>PERC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>3.977</td>
<td>1478</td>
<td>5040</td>
<td>1594</td>
<td>1022</td>
<td>5.3</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>6.655</td>
<td>2546</td>
<td>8158</td>
<td>2720</td>
<td>1382</td>
<td>27.2</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>10.080</td>
<td>4027</td>
<td>12366</td>
<td>4248</td>
<td>1691</td>
<td>7.2</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>14.747</td>
<td>6032</td>
<td>17563</td>
<td>6362</td>
<td>2104</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Table 14.1. Results obtained after testing SDSTEP on the STIFF-TEST problems.

* SDSTEP did not solve problem F5 (the iteration matrix becomes singular during the interval of integration).

As a comparison we list the results obtained by testing two codes GEAR and EPISODE (MF=21) based on backward differentiation formulas on the same computer.

† given in units of the tolerance.
### Table 14.2 Results obtained by testing GEAR on the STIFF DETEST problems

<table>
<thead>
<tr>
<th>Class</th>
<th>Time</th>
<th>Steps</th>
<th>FCN Calls</th>
<th>JAC. Calls</th>
<th>LU Decomp</th>
<th>MAXM. LOC:ERR.</th>
<th>PERC. DEC.</th>
<th>B.DEC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3.090</td>
<td>2988</td>
<td>3742</td>
<td>376</td>
<td>376</td>
<td>35.3</td>
<td>6.1</td>
<td>0.1</td>
</tr>
<tr>
<td>B</td>
<td>12.768</td>
<td>13319</td>
<td>14844</td>
<td>887</td>
<td>887</td>
<td>4.6</td>
<td>9.9</td>
<td>0.0</td>
</tr>
<tr>
<td>C</td>
<td>3.301</td>
<td>5131</td>
<td>6539</td>
<td>514</td>
<td>514</td>
<td>7.8</td>
<td>7.0'</td>
<td>0.1</td>
</tr>
<tr>
<td>D</td>
<td>1.686</td>
<td>2813</td>
<td>4302</td>
<td>418</td>
<td>418</td>
<td>2.7</td>
<td>3:2</td>
<td>0.0</td>
</tr>
<tr>
<td>E</td>
<td>2.159</td>
<td>3770</td>
<td>4939</td>
<td>439</td>
<td>439</td>
<td>17.7</td>
<td>7.8</td>
<td>0.1</td>
</tr>
<tr>
<td>F</td>
<td>6.331</td>
<td>9939</td>
<td>13350</td>
<td>975</td>
<td>975</td>
<td>10.5</td>
<td>9.4</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Summary**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29.335</td>
<td>37960</td>
<td>47712</td>
<td>2609</td>
<td>3609</td>
<td>35.3</td>
<td>8.3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tol</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>1.635</td>
<td>2343</td>
<td>3618</td>
<td>454</td>
<td>454</td>
<td>10.5</td>
<td>14.0</td>
<td>0.2</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>5.132</td>
<td>6768</td>
<td>8946</td>
<td>751</td>
<td>751</td>
<td>35.3</td>
<td>12.9</td>
<td>0.1</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>7.677</td>
<td>10581</td>
<td>13430</td>
<td>1006</td>
<td>1006</td>
<td>5.3</td>
<td>8.5</td>
<td>0.0</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>14.892</td>
<td>18268</td>
<td>21718</td>
<td>1398</td>
<td>1398</td>
<td>7.8</td>
<td>5.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Table 14.3 Results obtained by testing EPISODE on the STIFF DETEST problems

<table>
<thead>
<tr>
<th>Class</th>
<th>Time</th>
<th>Steps</th>
<th>FCN Calls</th>
<th>JAC. Calls</th>
<th>LU Decomp</th>
<th>MAXM. LOC:ERR.</th>
<th>PERC. DEC.</th>
<th>B.DEC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3.497</td>
<td>2751</td>
<td>4029</td>
<td>593</td>
<td>593</td>
<td>12.4</td>
<td>10.8</td>
<td>0.9</td>
</tr>
<tr>
<td>B</td>
<td>20.980</td>
<td>15747</td>
<td>20496</td>
<td>1304</td>
<td>1304</td>
<td>16.4</td>
<td>40.9</td>
<td>0.5</td>
</tr>
<tr>
<td>C</td>
<td>4.413</td>
<td>4596</td>
<td>6902</td>
<td>749</td>
<td>749</td>
<td>26.6</td>
<td>17.8</td>
<td>1.0</td>
</tr>
<tr>
<td>D</td>
<td>2.172</td>
<td>2419</td>
<td>3992</td>
<td>659</td>
<td>659</td>
<td>7.4</td>
<td>15.9</td>
<td>0.4</td>
</tr>
<tr>
<td>E</td>
<td>3.133</td>
<td>3372</td>
<td>5439</td>
<td>633</td>
<td>633</td>
<td>3.7</td>
<td>24.0</td>
<td>0.0</td>
</tr>
<tr>
<td>F</td>
<td>8.595</td>
<td>9186</td>
<td>14804</td>
<td>1730</td>
<td>1730</td>
<td>18.7</td>
<td>17.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Summary**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43.242</td>
<td>37798</td>
<td>55662</td>
<td>5668</td>
<td>5668</td>
<td>26.6</td>
<td>26.3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tol</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>4.864</td>
<td>4075</td>
<td>5886</td>
<td>821</td>
<td>821</td>
<td>8.9</td>
<td>36.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>7.190</td>
<td>6585</td>
<td>9662</td>
<td>1182</td>
<td>1182</td>
<td>18.7</td>
<td>33.2</td>
<td>0.4</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>10.662</td>
<td>9832</td>
<td>15350</td>
<td>1567</td>
<td>1567</td>
<td>26.6</td>
<td>32.3</td>
<td>0.6</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>20.527</td>
<td>17306</td>
<td>24764</td>
<td>2098</td>
<td>2098</td>
<td>19.6</td>
<td>17.9</td>
<td>0.2</td>
</tr>
</tbody>
</table>

*EPISODE failed to solve problems D3 and E5 at tolerance $10^{-2}$. 

Technion - Computer Science Department - Technical Report CS0145 - 1979
In comparing the results for these methods one should bear in mind that the STIFF-DETEST problems are mostly small (the largest contains 10 equations) and mildly stiff (with stiffness ratios ≈ 10^4). Also the absolute error criterion which is used is not necessarily the most appropriate choice on some of the problems. The GEAR and EPISODE packages contain other options (for example, it is not necessary to supply the exact Jacobian) which may be more efficient or convenient to use on certain stiff problems. For another comparison of the performances of the GEAR and EPISODE packages, see for example Byrne et al [1977].

The results of Tables 14.1, 14.2 and 14.3 indicate that SDSTEP performs competitively with the other codes on these problems, and, as expected, becomes relatively more efficient than the other codes on those problems whose eigenvalues contain large imaginary parts (Class B) and at more stringent tolerances. Also the code SDSTEP was more accurate than the other two on these problems.

It should be pointed out that for each LU decomposition (which requires \(\frac{1}{3} n^3\) operations) an extra \(n^3\) operations are required to square the Jacobian when calculating the iteration matrix. (This is not the case for codes based on backward differentiation formulas.) Hence in solving large systems, advantage must be taken, when possible, of the structure of the Jacobian or a different approach to solving the linear systems should be adopted. As discussed in Section 5, several possibilities are available and the code has been designed so that these options can be easily incorporated.

Other future developments could include the facility to switch between Adams formulas and second derivative formulas when solving a stiff ODE. Because all the arrays used in the Adams code STEP are
common to both STEP and SDSTEP, it should be possible to switch between routines without starting and without extra calculation.

Of course, more testing is required. Some of the heuristics used in the code may be improved and others introduced. We hope we have facilitated this by clearly listing all the strategies used in the current version of the code.

ACKNOWLEDGEMENTS

Much of this work was done while I was visiting the University of Toronto and I would like to express my appreciation to Professors Wayne Enright and Tom Hull and to Dr. Ken Jackson for many stimulating discussions and for contributing many ideas.
REFERENCES


REFERENCES (cont'd)


C THIS IS A TEST PROBLEM FOR SDSTEP.
C
C INITIAL CONDITIONS: AT X = 0, Y(1) = 1, Y(2) = Y(3) = 0.
C
C INTEGRATE UNTIL X = 40 USING AN ABSOLUTE ERROR CRITERION. SOLVE
C THE PROBLEM AT FOUR DIFFERENT TOLERANCES 10^-2, 10^-4, 10^-6
C AND 10^-8.
C
C IMPLICIT REAL*8 (A-X,C-Z)
C DIMENSION Y(3),WT(3),PHI(3,10),YP(3),PODERIV(3,3),NBAR(3,3),
C NPIV(3)
C EXTERNAL F,PODERIV
C LOGICAL START,CRASH
C COMMON /NSTATS/,NFNS,NJACS,NLU
C
C SOLVE THE PROBLEM AT FOUR DIFFERENT TOLERANCES STARTING AT 1.0.0-2.
C
C EPS = 1.0
C DO 995 IX = 1,4
C EPS = EPS*1.0-2
C WRITE (6,10) EPS
C 10 FORMAT(1HO,12HTOLERANCE = ,D10.4)
C NFNS = 0
C NJACS = 0
C NLU = 0
C X = 0.0
C Y(1) = 1.0
C Y(2) = 0.0
C Y(3) = 0.0
C NEQN = 2
C T = 1.0
C WT(1) = 1.0
C WT(2) = 1.0
C WT(3) = 1.0
C START = .FALSE.
C XEND = 40.0
C ICCLNT = 0
C
C CONTINUE
C IF (M = GT. XEND - X) M = XEND - X
C CALL SDSTEP(X,Y,F,PODERIV,NEQN+F,EPS,WT,START,HCLD,K,KOLD,
C CRASH,PHI,YP,PODERIV,NEAR,NPIV)
C ICOUNT = ICOUNT + 1
C IF (CRASH) GO TO 50
C IF (X = LTE. XEND) GO TO 20
C WRITE(6,30) X,(Y(I),I = 1,3)
C 30 FORMAT(1H0,1X,15I6)
C WRITE (6,40) ICCLNT,NFNS,NJACS,NLU
C 40 FORMAT(1H0,6HSTEPS,14,5H FNS,14,6H JACS,14,6H NLU,14)
C MJ T5 = 999
C 50 WRITE(6,00)
C
See output on page 56.
SUBROUTINE SDSTEP (X,Y,F,PDERV,NECN,H,EPS,T,START,HOLD,K,KOLD,
1 CRASH,PHI,YP,PDERIV,WEAR,NPIV)

SUBROUTINE SDSTEP INTEGRATES A SYSTEM OF UP TO 20 STIFF FIRST
ORDER ORDINARY DIFFERENTIAL EQUATIONS ONE STEP NORMALLY FROM X
TO X + H, USING A FIXED LEAGING COEFFICIENT IMPLEMENTATION OF
ENRIGHT'S SECOND DERIVATIVE FORMULAS.

THE PARAMETERS REPRESENT:
X -- THE INDEPENDENT VARIABLE.
Y(*) -- THE SOLUTION VECTOR AT X.
NECN -- THE NUMBER OF EQUATIONS TO BE INTEGRATED.
H -- THE APPROPRIATE STEPSIZE FOR THE NEXT STEP. IT IS NORMALLY
DETERMINED BY THE CODE.
EPS -- THE LOCAL ERROR TOLERANCE (MUST BE VARIABLE).
T(*) -- A VECTOR OF WEIGHTS FOR THE ERROR CRITERION.
START -- A LOGICAL VARIABLE SET .TRUE. FOR THE FIRST STEP
AND .FALSE. OTHERWISE.
HOLD -- THE STEPSIZE USED ON THE LAST SUCCESSFUL STEP.
K -- THE APPROPRIATE ORDER FOR THE NEXT STEP (DETERMINED BY CODE).
KOLD -- THE ORDER USED ON THE LAST SUCCESSFUL STEP.
CRASH -- A LOGICAL VARIABLE SET .TRUE. WHEN KC STEP CAN BE
TAKEN AND .FALSE. OTHERWISE.
YP(*) -- DERIVATIVE OF THE SOLUTION AT X AFTER A SUCCESSFUL STEP.
PDERIV(*) -- PDERIV(I,J) IS THE APPROXIMATION TO THE (I,J)
ELEMENT OF THE JACOBIAN DF/DO AFTER A
SUCCESSFUL STEP.

THE ARRAY PHI CONTAINS THE MODIFIED DIVIDED DIFFERENCES AND THE
ARRAYS WEAR AND NPIV ARE USED BY THE LINEAR ALGEBRA Routines.

INPUT TO SDSTEP

FIRST CALL -

THE USER MUST PROVIDE STORAGE IN HIS DRIVER PROGRAM FOR ALL ARRAYS
IN THE CALL LIST; NAMELY

DIMENSION Y(NECN),T(NECN),PHI(NECN,10),YP(NECN),
PDERIV(NECN,NECN),WEAR(NECN,NECN),NPIV(NECN)

THE USER MUST ALSO DECLARE START AND CRASH LOGICAL VARIABLES
AND F AND PDERV EXTERNAL SUBROUTINES, SUPPLY THE SUBROUTINES
F(X,Y,YP) TO EVALUATE
YP(I) = F(Y(1),Y(2),....,Y(NECN))
AND PDERIV(X,Y,PDERIV,NECN) TO EVALUATE
PDERIV(I,J) = PARTIAL CF{SUB I)/SY (SUB J)

AND INITIALIZE THE FOLLOWING PARAMETERS ONLY:
X -- THE INITIAL VALUE OF THE INDEPENDENT VARIABLE.
Y(*) -- THE VECTOR OF INITIAL VALUES OF DEPENDENT VARIABLES.
NECN -- THE NUMBER OF EQUATIONS TO BE INTEGRATED.
H -- A NOMINAL STEPSIZE INDICATING THE DIRECTION OF INTEGRATION
AND THE MAXIMUM SIZE OF THE FIRST STEP. MUST BE VARIABLE.
EPS -- THE LOCAL ERROR TOLERANCE PER STEP. MUST BE VARIABLE.
T(*) -- A VECTOR OF NECN-2EARCH WEIGHTS FOR THE ERROR CRITERION.
START -- .TRUE.

SDSTEP REQUIRES THE L2 NECN OF THE VECTOR WITH COMPONENTS
LOCAL ERROR(NECN)/T(NECN) TO BE LESS THAN EPS FOR A SUCCESSFUL STEP.
THE ARRAY T ALLOWS THE USER TO SPECIFY AN ERROR TEST.
• THE ONLY MACHINE DEPENDENT
• ROUNDOFF ERROR U WHICH IS THE SMALLEST POSITIVE NUMBER SUCH THAT
• * 1.0+U GT 1.0 * THE USER MUST CALCULATE U AND INSERT T/DU=2.0*U *
C * AND FOURU=M40U IN THE CALL STATEMENT BEFORE CALLING THE CODE. * 00001230
DATA TWOU,FORU, */ 00001240
C********************************************************************************************** 00001250
DATA PSTAR/0.0,0.167E0,0.139E-1,0.486E-2,0.236E-2,0.136E-2, 00001260
1.0.663E-3,0.590E-3,0.424E-3,0.317E-3/ 00001270
DATA ICOEFF/1.2,2.9,307,3133,3177,2,247021,115E623,1.3,48,540, 00001280
1,5760.6,64,8000,483840,3228000,-1.9,-1.9,-1.9,-1.9,-863, 00001290
1,-275,.33553,2.6,8,1.E6,32,1.0680,34554,453600/ 00001300
DATA G(1),G(2)/1.0,0.5/, SIGU=1.0 00001310

*** BEGIN BLOCK 0 *** 00001320
CHECK IF THE STEPSIZE OR ERROR TOLERANCE IS TOO SMALL FOR MACHINE PRECISION. IF FIRST STEP, INITIALIZE PHI ARRAY AND ESTIMATE A STARTING STEPSIZE. 00001330
*** 00001340
IF THE STEPSIZE IS TOO SMALL, DETERMINE AN ACCEPTABLE ONE. 00001350

CRASH = .TRUE. 00001360
IF (DABS(H) *GE. FOURU) GC TO 5 00001370
H = DSIGN(FOURU,DABS(X),H) 00001380
RETURN 00001390
5 PSEPS = 0.5*EPS 00001400

IF THE ERROR TOLERANCE IS TOO SMALL, INCREASE IT TO AN ACCEPTABLE VALUE. 00001410
ROUND = 0.0 00001420
DO 10 L = 1,NEQN 00001430
10 ROUND = ROUND + (Y(L)/HT(L))**2 00001440
RETURN 00001450
15 CRASH = .FALSE. 00001460
IF (.NOT. START) GO TO 99 00001470

C INITIALIZE. 00001480
CALL F(X,Y,YP) 00001490
CALL PDERIV(X,Y,PDERIV,NECN) 00001500

C TAKE A TRIAL STEP OF MAGNITUDE SQRT(FOURU) TO OBTAIN AN APPROXIMATION TO Y'**(x). THIS VALUE IS USED TO ESTIMATE AN APPROPRIATE STEPSIZE FOR THE FIRST STEP. 00001510

C ABSH = DABS(H) 00001520
HTEMP = DSIGN(DSORT(FCLC),H) 00001530
DO 25 L = 1,NEGN 00001540
DD(L) = 0.0 00001550
DO 20 J = 1,NEGN 00001560
20 DD(L) = DD(L) + PDERIV(J)*Y(J) 00001570
25 YTRIAL(L) = Y(L) + HTEMP*YP(L) + 0.5*HTEMP*DD(L) 00001580
CALL F(X+HTEMP,YTRIAL,YPSAVE) 00001590

C ESTIMATE THE (WEIGHTED) ACF OF Y'**(X)/6. 00001600

C SUM = 0.0 00001610
DO 30 L = 1,NEGN 00001620
TEMP1 = YPSAVE(L) - YP(L) - HTEMP*DD(L) 00001630

Technion - Computer Science Department - Technical Report CS0145 - 1979
30 SUM = SUM + (TEFL/WT(L))**2
SUM = DSQRT(SUM)/(3.0*TEMP*HTEMP)
C
C CALCULATE THE INITIAL STEPSIZE.
C
IF (PSEPS *LT, 6.0*SUM*AESH*3) AESH = 0.5*(PSEPS/SUM)**(1.0/3.0)
H = DSIGN(DMAXI(AESH,FCURK*PABS(X)*H)
DO 35 L = 1,NECN
PHI(L,1) = YP(L)
35 PHI(L,2) = H*DD(L)
HOLD = 0.0
HPAST = 0.0
K = 2
KOLD = 0
PSI(I) = h
RAX = 2.0
START = .FALSE.
PHASEI = .TRUE.
99 IFAIL = 0
*** END BLOCK 0 ***
C
*** BEGIN BLOCK 1 ***
C COMPUTE COEFFICIENTS OF FORMULAS FOR THIS STEP. AVOID COMPUTING
THOSE QUANTITIES NOT CHANGED WHEN THE STEPSIZE IS NOT CHANGED.
C
C
100 KPI = K + 1
KP2 = K + 2
KM1 = K - 1
KM2 = K - 2
C
C NS IS THE NUMBER OF STEPS TAKEN WITH SIZE H, INCLUDING THE CURRENT
ONE. WHEN K = LT NS, NO COEFFICIENTS CHANGE.
C
IF (H .NE, HPAST) NS = 0
NS = NS+1(KOLD+1)
NSP1 = NS + 1
IF (K .LT, NS) GO TO 199
C
C COMPUTE THOSE COEFFICIENTS OF ALPHA(*),BETA(*),PSI(*),SIG(*) WHICH
HAVE CHANGED.
C
BETA(NS) = 1.0
REALNS = NS
ALPHA(NS) = 1.0/REALNS
TEMP1 = H*REALNS
SIG(NSP1) = 1.0
IF (K .LT, NSP1) GO TO 110
DO 105 I = NSP1,K
IMI = I - 1
TEMP2 = PSI(IMI)
PSI(IMI) = TEMP1
BETA(I) = BETA(IMI)*FSI(IMI)/TEMP2
TEMP1 = TEMP2 + H
ALPHA(I) = H/TEMP1
REALI = I
105 SIG(I+1) = ALPHA(I)*SIG(I)
C
C
C
Technion - Computer Science Department - Technical Report CS0145 - 1979
110 PSII(K) = TEMP1
C
C COMPUTE CGEFFECTS G(*).
C
C INITIALIZE V(*) AND W(*) - G(2) IS SET IN DATA STATEMENT.
C
IF (NS + GT + 1) GO TO 120
DO 115 IQ = 1, K
TEMP3 = IQ*(IQ+1)
V(IQ) = 1.0/TEMP3
115 W(IQ) = *V(IQ)
GO TO 140
C
C IF ORDER WAS RAISED, UPDATE DIAGONAL PART OF V(*).
C
120 IF (K LE KOLD) GO TO 130
TEMP4 = K*KPI
V(K) = 1.0/TEMP4
NSM2 = NS - 2
IF (NSM2 LT 1) GO TO 130
DO 125 J = 1, NSM2
125 I = K-J
125 V(I) = V(I) - ALPHA(J+1)*V(I+1)
C
C UPDATE V(*) AND SET W(*).
C
130 LIMIT1 = KPI - NS
TEMP5 = ALPHA(NS)
DO 135 IQ = 1, LIMIT1
V(IQ) = V(IQ) - TEMP5*V(IQ+1)
135 W(IQ) = V(IQ)
G(NSPI) = W(I)
C
C COMPUTE THE G(*) IN THE WORK VECTOR *W(*).
C
140 NSP2 = NS + 2
IF (KPI LT NSP2) GO TO 199
DO 150 I = NSP2, KPI
LIMIT2 = KPI - I
TEMP6 = ALPHA(I-1)
DO 145 IQ = 1, LIMIT2
145 V(IQ) = V(IQ) - TEMP6*V(IQ+1)
150 W(I) = W(I)
CONTINUE
199 CONTINUE
*** END BLOCK 1 ***
C
C *** BEGIN BLOCK 2 ***
C PREDICT THE SOLUTION AND ITS DERIVATIVES.
***
C
C CHANGE PHI TO PHI STAT.
C
IF (K LT NSPI) GO TO 215
DO 210 I = NSPI, K
TEMP1 = BETA(I)
DO 205 L = 1, NCHA
205 PHI(L+I) = TEMP1*PHI(L+1)

210 CONTINUE
C PREDICT THE SOLUTION, P(1), AND ITS DERIVATIVES.
C
215 DO 220 L = 1, NEGN
    P(L) = 0.0
    PD(L) = 0.0
220  HPDL(L) = 0.0
    ALPBAR = 0.0
    DO 230 I = 1, K
    DO 225 L = 1, NEGN
    TEMP2 = PHI(L, I)
    P(L) = P(L) + G(I)*TEMP2
    PD(L) = PD(L) * TEMP2
225  HPDL(L) = HPDL(L) + ALPBAR*TEMP2
230  ALPBAR = ALPBAR + ALPHA(I)
    DO 245 L = 1, NEGN
    245  P(L) = Y(L) + H*P(L)
    XOLD = X
    X = X + M
    ABSH = DABS(H)
C *** END BLOCK 2 ***

C *** BEGIN BLOCK 3 ***
C SOLVE THE CORRECTOR EQUATION USING A MODIFIED NEWTON-RAPHSON
C SCHEME.
C ***
C CONVGD = .TRUE.
C SET THE LEADING COEFFICIENTS OF THE CORRECTOR EQUATION.
C
    TEMP1 = ICOEFF(K-1)
    TEMP2 = ICOEFF(K+7)
    TEMP3 = ICOEFF(K+15)
    TEMP4 = ICOEFF(K+23)
    HKETA = (H*TEMP1)/TEMP2
    HGAMMA = (H*TEMP3)/TEMP4
    HSQKAP = H*HGAMMA
    HSQKP1 = CABS(H*G(KFI))
C DETERMINE THE STARTING VALUE FOR THE NEWTON ITERATION SCHEME.
C
    IF (K .EQ. 2 .AND. .NOT. PHASE1) GC TO 310
    DO 305 L = 1, NEGN
    305  YTRIAL(L) = P(L)
    GO TO 320
    310  TEMPS = H/HOLD
    DO 315 L = 1, NEGN
        315  YTRIAL(L) = Y(L) + TEMPS*(Y(L) - YAM1(L))
    320  CALL F(X, YTRIAL, YP)
        CALL PDHV(X, YTRIAL, PDERV, NEGN)
C CALCULATE THE NORM OF THE STARTING VALUE AND THE CONSTANT
C PART OF THE CORRECTOR EQUATION.
C
    00003060 00003070 00003080 00003090 00003100 00003110 00003120 00003130 00003140 00003150 00003160 00003170 00003180 00003190 00003200 00003210 00003220 00003230 00003240 00003250 00003260 00003270 00003280 00003290 00003300 00003310 00003320 00003330 00003340 00003350 00003360 00003370 00003380 00003390 00003400 00003410 00003420 00003430 00003440 00003450 00003460 00003470 00003480 00003490 00003500 00003510 00003520 00003530 00003540 00003550 00003560 00003570 00003580 00003590 00003600 00003610 00003620 00003630 00003640 00003650 00003660 00003670 00003680 00003690 00003700 00003710

Technion - Computer Science Department - Technical Report CS0145 - 1979
YNORM = 0.0
DO 325 L = 1,NECN
  YNORM = YNORM + (YTRIAL(L)/WT(L))**2
325 C(L) = P(L) - MBETA*PD(L) - HGAP**HPPDL(L)
YNORM = DSQRT(YNORM)
YNORM = 1.E-10
C
ON THE FIRST ITERATION, YBAR IS RE-EVALUATED IF THE STEPSIZE
OR ORDER HAS CHANGED.
C
ICALC = 0
IF (K .EQ. KOLD .AND. NS .GT. 1) GC TO 335
C
USING THE SUBROUTINE WCALC, RE-EVALUATE YBAR IN FACTORED FORM.
C SET FLAG TO INDICATE THAT YBAR HAS BEEN RE-EVALUATED.
C
330 ICALC = 1
  CALL WCALC(NECN,YBAR,PDERIV,MBETA,HSGAP,PIV,SINGLR)
C
CHECK THAT YBAR IS NOT SINGULAR.
C
IF (SINGLR) GO TO 375
C
SET THE ITERATION COUNTER.
C
335 M = 0
340 M = M + 1
C
CALCULATE THE RESIDUAL AND STORE IT IN YD(*).. 
C
DO 350 L = 1,NECN
  TEMP6 = 0.0
  DO 340 J = 1,NECN
    TEMP6 = TEMP6 + PDERIV(L,J)*VF(J)
 345 TEMP6 = TEMP6 + PDERIV(L,J)*VF(J)
350 YD(L) = -(YTRIAL(L) - MBETA*YP(L) - HSGAP*TEMP6 - C(L))
C
USING THE SUBROUTINE WSOLVE, CALCULATE YD, THE DIFFERENCE
BETWEEN THE NEXT AND CURRENT APPROXIMATIONS TO Y(A+I).
C
CALL WSOLVE(NECN,YBAR,PDERIV,PIV,YD)
YDSAVE = YDONLY
YDCRFL = 0.0
DO 355 L = 1,NECN
355 YDONLY = YDONLY + (YD(L)/WT(L))**2
YNORM = DSQRT(YDCRFL)
C
IN THE FOLLOWING SECTION THE CONVERGENCE TESTS ARE MADE.
C
IF (YNORM .LE. FCURU*YNORM) GO TO 380
DO 360 L = 1,NECN
  YTRIAL(L) = YTRIAL(L) + YD(L)
360 YPSAVE(L) = YP(L)
  CALL F(X,YTRIAL,YP)
  FSAVE = FDONLY
  FDONLY = 0.0
  DO 365 L = 1,NECN
365 FDONLY = FDONLY + ((YP(L)-YPSAVE(L))/WT(L))**2
  FDONLY = HGAP**DSQRT(FDONLY)
  IF (YNORM .LE. 0.1*EPS .AND. FDONLY .LE. 0.1*EPS) GO TO 380
C
ANOTHER ITERATION IS REQUIRED. CHECK THAT THE SCHEME IS
CONVERGING AND THAT THE NUMBER OF ITERATIONS WITH THE CURRENT ITERATION MATRIX DOES NOT EQUAL 4.

CRATE = YDNORM/VDSAVE

IF (CRATE .GE. 1.0) GO TO 370
IF (M .LT. 4) GO TO 340

NO CONVERGENCE IN 4 ITERATIONS. CHECK WHETHER IT IS WORTH TRYING ANOTHER ITERATION.

FCRATE = FDNorm/FCSAVE

IF (M .EQ. 4 .AND. CMAX1(YDNORM*CRATE,FDNorm*FCRATE) .LE. 0.1*EPS) GO TO 340

NO CONVERGENCE WITH THE CURRENT ITERATION MATRIX. IF WEAR HAS NOT BEEN RE-EVALUATED THIS STEP, THEN RE-EVALUATE MBAR AND TRY ANOTHER 5 ITERATIONS. OTHERWISE SET CONVGD EQUAL TO .FALSE. INDICATING THE NEWTON SCHEME DID NOT CONVERGE.

370 CONTINUE

IF (J_CALC .EQ. 0) GO TO 330

379 CONVGD = .FALSE.

CHECK IF THE NEWTON SCHEME CONVERGED.

380 IF (.NOT. CONVGD) GC TC 500

*** END BLOCK 3 ***

*** BEGIN BLOCK 4 ***


***


400 ERK = 0.0
Erkm1 = 0.0
Erkm2 = 0.0
DO 405 L = 1,NEG

Temp1 = X(l) - PF(l)

Erk = ERK + TEMP1*TEMP1
IF (K .LE. 2) GO TO 405
Temp2 = TEMP1 + Phi(L,K)

Erkm1 = ERKM1 + TEMP2*TEMP2
IF (K .LE. 3) GC TO 405

Temp3 = TEMP2 + Phi(L,KM1)

Erkm2 = ERKM2 + TEMP3*TEMP3

405 CONTINUE

IF (K-3) 420,415,410

410 Erkm2 = ABsh*Sig(KM1)*Fistar(KM2)*CSQRT(ERKM2)
415 Erkm1 = ABsh*Sig(K)*Fistar(KM1)*CSQRT(ERKM1)
420 Erk = ABsh*Sig(KP1)*Fistar(K)*DSQRT(ERK)

CALCULATE LOCAL ERRORS AT CFER K.

Temp4 = H*G(KP1)
DO 425 L = 1, NEQN
425 CD(L) = TEMP*(YP(L) - PD(L)) - (YTRIAL(L) - P(L))
CALL WSOLVE(NEQN,NEK*,APIV,DD)
ERR = 0.0
DO 430 L = 1, NEQN
430 ERR = ERR + (DD(L)/Y(L))**2
ERR = DSQRT(ERR)
KNEW = K
C TEST IF THE ORDER SHOULD BE LOWERED.
C IF (K - 3) 445, 440, 435
435 IF (DMAX1(ERKM1,ERKM2) < EPS) KNEW = K1
GO TO 445
440 IF (ERKM1 <= 0.5*ERK) KNEW = K1
C TEST IF THE STEP IS SUCCESSFUL.
C 445 IF (ERR <= EPS) GO TO 600
C *** END BLOCK 4 ***
C *** BEGIN BLOCK 5 ***
C THE STEP IS UNSUCCESSFUL. RESTORE X AND PSI IF THIRD
C CONSECUTIVE FAILURE, SET ORDER TO 2. IF STEP FAILS MORE THAN
C THREE TIMES REDUCE STEPSIZE BY A FACTOR OF 10. DOUBLE THE ERROR
C TOLERANCE AND RETURN IF THE NEW STEPSIZE IS TOO SMALL FOR
C MACHINE PRECISION.
C ***
C RESTORE X, PSI(*) AND PHI(*).
C 500 PHASE1 = .FALSE.
X = XOLD
DO 505 I = 2, K
505 PSI(I-1) = PSI(I) - H
IF (K < LT. NSP1) GO TO 520
DO 515 I = NSP, K
TEMP1 = 1.0/BETA(I)
DO 510 L = 1, NEGN
510 PH1(L,I) = TEMP1*PH1(L,I)
515 CONTINUE
C DETERMINE THE APPROPRIATE STEPSIZE AND ORDER FOR CONTINUING
C THE INTEGRATION.
C 520 HPAST = H
IFAIL = IFAIL + 1
IF (IFAIL GT 10) GO TO 530
C FIRST FAILURE THIS STEP, CHECK CAUSE OF FAILURE.
C IF (CONVGO) GO TO 625
C HERE CONVGD = .FALSE. INDICATING THAT THE CAUSE OF THE
C FAILURE WAS THE NON-CONVERGENCE OR THE NEWTON ITERATION
C SCHEME.
HERE CONVG = *TRUE* INDICATING THAT THE NEWTON SCHEME
CONVERGED AND THAT THE CAUSE OF THE FAILURE WAS THE ERROR
ESTIMATE EXCEEDING THE TOLERANCE.

525. TEMP2 = K + 1
   R = 0.95*(ERR/PSEPS + .0001)**(-1./6./TEMP2)
   TEMP3 = 0.1
   TEMP4 = 0.9
   R = DMAX(TEMP3,DMIN1(TEMP4,R))
   K = KNEW
   GO TO 545

MULTIPLE FAILURES. REDUCE THE STEPSIZE BY A
FACTOR OF 4. IF FAILURE CAUSED BY THE ERROR ESTIMATE BEING
TOO LARGE, CONSIDER DECREASE IN ORDER. ON THIRD FAILURE SET
ORDER TO 2. THEREAFTER REDUCE STEPSIZE BY A FACTOR OF 10.

530 R = 0.25
   IF (CONVG = K = KNEW)
   IF (IFAIL = 3) 545, 540, 535
535 R = 0.1
540 K = 2
545 H = HMR
   IF (DABS(H) .GE. FCURU.CAES(X)) GO TO 550
   CRASH = *TRUE*.
   H = DSIGN(FCURU.CAES(X),H)
   EPS = EPS + EPS
   RETURN
550 GO TO 100
*** END BLOCK 5 ***

*** BEGIN BLOCK 6 ***
THE STEP IS SUCCESSFUL. DETERMINE THE EST ORDER AND STEPSIZE
FOR THE NEXT STEP. UPDATE THE DIFFERENCES FOR THE NEXT STEP.
***

600 KDIFF = K - KOLD
   KOLD = K
   HOLD = H
   Hpast = H

ESTIMATE THE ERROR AT ORDER K+1 UNLESS:
ALREADY DECIDED TO LOWER ORDER,
STEPSIZE NOT CONSTANT,
ORDER RAISED IN PREVIOUS STEPS.

?ERKPI = 0.0
   IF (KNEW .EQ. K1) .AND. K .EQ. 9) FPASE1 = *FALSE*.
   IF (FPASE1) GO TO 615
   IF (KNEW .EQ. K1) GO TO 620
   IF (KP1 .GT. NS) OR. KDIFF .EQ. 1) GO TO 625
   DD 605 L = 1,NEQN
   *TEMP1 = YP(L) - FC(L) - PHI(L,KPI)

605  ERKPI = ERKPI + TEMP1*TEMP1

Technion - Computer Science Department - Technical Report CS0145 - 1979
ERKPI = ABSP*PISTAR(KF1)**DSQRT(ERKF1)

IF (K * GT. 2) GO TO 610
IF (ERKPI * LE. 0.5*ERK) GO TO 625
GO TO 615

610 IF (ERKMI * LE. DMIN1(ERK,ERKPI)) GO TO 620
IF (ERKPI * GE. ERK * QF. K * EQ. '9') GO TO 625

HERE ERKPI * LT. ERK * LT. MAX(ERKMI,ERKMI) ELSE THE ORDER WOULD HAVE BEEN LOWERED IN BLOCK 4. THIS ORDER IS TO BE RAISED.

FAKE ORDER.

615 K = KPI
GO TO 625

LOWE ORDER.

620 K = KPI

DETERMINE THE APPROPRIATE STEPSIZE FOR THE NEXT STEP.

625 MNXW = 'H

TEMP2 = KOLD + 1
.R = 0.95*(ERK/PSEPS + 0.0001)**(-1.C/TEMP2)
IF (.R * LT. 1.5) GO TO 630

MNEW = H*DMIN1(R,R*X).

630 'H = MNEW

UPDATE YMI AND Y;

DO 635 L = 1,NEQN
    YMI(L) = Y(L)
635

UPDATE DIFFERENCES FOR THE NEXT STEP.

LIMI = KOLD + 1
CD 640 L = 1,NEQN

DO 640 I = 1,KOLD
    J = LIMIT - 1
    DO 645 L = 1,NEQN

645 PHI(L,J) = PHI(L,J) + PHI(L,J+1)

CONTINUE

RETURN

END

*** END BLOCK 6. ***
SUBROUTINE WCALC (NEQN,WBAR,POERIV,HBETA,HSQGAM,NPIV,SINGLR)  

C THIS SUBROUTINE CALCULATES THE ITERATION MATRIX, WBAR, USED IN THE NEWTON-RAPHSON SCHEME TO SOLVE THE CORRECTOR EQUATION. WBAR IS CALCULATED IN FACTORED LU FORM. IF WBAR IS SINGULAR A RETURN IS MADE IN SINGLR.

DIMENSION POERIV(NEQN,NEQN),WBAR(NEQN,NEQN),NPIV(NEQN)  
DIMENSION WORK(20)  
LOGICAL SINGLR

C FORM MATRIX WBAR IN UNFACTORED FORM.

DO 15 I = 1,NEQN
  DO 10 J = 1,NEQN
    WBARI(I,J) = 0.0
  DO 5 M = 1,NEQN
    5 WBARI(I,J) = WBARI(I,J) + POERIV(I,M)*POERIV(M,J)
  10 WBARI(I,J) = -HSQGAM*WBARI(I,J)
  15 CONTINUE

DO 25 I = 1,NEQN
  DO 20 J = 1,NEQN
    WBARI(I,J) = WBARI(I,J) - HBETA*POERIV(I,J)
  25 CONTINUE

DO 30 J = 1,NEQN
  30 WBARI(I,I) = WBARI(I,I) + 1.0.

C CALL DECOMP(NEQN,NEQN,WBAR,COND,NPIV,WORK)

C CHECK TO SEE IF WBAR IS SINGULAR.

SINGLR = .FALSE.
CONDPI = COND + 1
IF (CONDPI .EQ. COND) SINGLR = .TRUE.
RETURN

END

SUBROUTINE WSOLVE (NEQN,WBAR,NPIV,B)  

C THIS SUBROUTINE SOLVES THE LINEAR SYSTEM OF EQUATIONS WBAR*X = B

IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION WBAR(NEQN,NEQN),NPIV(NEQN),B(NEQN)
CALL SOLVE(NEQN,NEQN,WBAR,B,NPIV)
RETURN
END

- 55 -
TOLERANCE = 0.10000-01

X 0.4000000000 02 Y(*) 0.71583284715416456 00 0.91657593485815700-01 0.284157967046468620 02
STEPS 35 FNS 142 JACS 36 NLU 34

TOLERANCE = 0.10000-03

X 0.4000000000 02 Y(*) 0.71582651541063780 00 0.91655132602995600-01 0.284164295076046800 02
STEPS 62 FNS 238 JACS 64 NLU 49

TOLERANCE = 0.10000-05

X 0.4000000000 02 Y(*) 0.71582767607621730 00 0.91655350503800200-01 0.284163738385372900 02
STEPS 103 FNS 400 JACS 111 NLU 75

TOLERANCE = 0.10000-07

X 0.4000000000 02 Y(*) 0.71582706331585070 00 0.91655347605348600-01 0.284163745845346200 02
STEPS 148 FNS 533 JACS 153 NLU 81