27
9-27-77
25
SAND77-1129
Unlimited Release

3151

MASTER

Solving Ordinary Differential Equations in SPEAKEASY

Lawrence S. Shampine

Prepared by Sandia Laboratories, Albuquerque, New Mexico 87115
and Livermore, California 94550 for the United States Energy Research
and Development Administration under Contract AT(29-1)-789
Printed August 1977

Sandia Laboratories

SF 2900 Q(7-73)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Issued by Sandia Laboratories, operated for the United States Energy Research & Development Administration by Sandia Corporation.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research & Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

SAND 77-1129
Unlimited Release
Printed August 1977

Solving Ordinary Differential Equations in SPEAKEASY

Lawrence F. Shampine
Numerical Mathematics Division
Sandia Laboratories
Albuquerque, New Mexico 87115

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
Price: Printed Copy ~~\$4.00~~; Microfiche \$2.25
4.50

Solving Ordinary Differential Equations in SPEAKEASY

L. F. Shampine, 5122

1. Introduction

SPEAKEASY [1] is a very high level language designed to make using computers easy for model building. This document describes the results of an investigation made to provide a capability for solving differential equations in the language. Although SPEAKEASY is a scientific tool seeing significant use, this is not our main reason for the study. The language is representative of situations in which one wants to regard the solution of a differential equation as a high level operation requiring no user intervention at all. This kind of operation is beyond the state of the art at this time because present codes require significant user input, and possibly intervention, and because present codes (with at least the exception [2]) do not even try to estimate error in the sense expected by the unsophisticated user, much less control it. The situation is in part due to considerations of efficiency, it being too expensive to achieve some desirable objectives in general scientific computation. In the context of SPEAKEASY the problems are relatively inexpensive compared to those of general library usage, and the deliberate decision has been made to pay any reasonable cost to keep computation as simple and as reliable as possible for the user. Thus the language affords a natural setting in which to find out how far the boundaries of software for ordinary differential equations can be extended.

Without going into the nature of SPEAKEASY the reader will not be able to see the origin of the difficulty, but the fact is that the

differential equations to be solved are only defined at certain points. Thus it is necessary to confront the basic question of solving differential equations when it is not possible to evaluate the equation anywhere we like. This is completely different from the task presented library codes, though it is of frequent occurrence due to coefficients which are measured data. Investigation of this matter also helps advance the state of the art of solving ordinary differential equations.

The algorithms and the code realizing them do quite well at meeting the objectives posed. Examples given in the last section will allow the reader to judge the success for himself. The principal remaining task is to cope with stiff differential equations. The present version detects such problems and the user is informed that the code is unsuitable. We plan to extend the code so as to integrate efficiently stiff problems but this task will require the resolution of some difficult questions. A number of aspects of the present code design were treated in a way which will facilitate these further developments.

2. The Problem

SPEAKEASY does not have within it functions in the sense of FORTRAN and other general purpose programming languages. A function like $\sin(x)$ can be employed in SPEAKEASY but it is defined as a table of values on a specified set of points and it is held in the computer as an array. For this reason we cannot even talk about the solution of a general problem like

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

because the function f would have to be defined on all the necessary values of y when the problem is posed within the language, and these

values are not known. The most general problem that can be readily described in SPEAKEASY is the class of linear problems

$$\underline{y}' = a(x)\underline{y} + \underline{g}(x)$$

where

$$a(x) = (a_{ij}(x)) \quad 1 \leq i, j \leq n$$

$$\underline{g}(x) = (g_i(x)) \quad 1 \leq i \leq n$$

$$\underline{y}(a) \text{ given, integrate from } x = a \text{ to } x = b$$

We note that one can write codes to solve other special classes of problems such as

$$y' = a(x)y^m + g(x)$$

because the subroutine used by the language can be written in FORTRAN and it is a question then of what kinds of functions we wish to make available. In this document we are going to consider only the general linear problem.

When the user defines the problem he provides the matrix $a(x)$ and the vector $\underline{g}(x)$. To do this he had to specify the arguments x . By this action he has implicitly defined where the solution $\underline{y}(x)$ is to be evaluated. Because the user is unlikely to have good information about where it is appropriate to evaluate $\underline{y}$ and because equal spacing in x is convenient, we shall suppose this to be the case. Thus we presume that the problem is to be integrated from XBEGIN to XEND using M equally spaced points. This corresponds to a spacing, or step size, $H = (XEND - XBEGIN)/(M-1)$. The user defines within SPEAKEASY arrays Y, G, A which for a system of N equations are:

Y is an N x M array of solution components

$$Y(I,K) \doteq y_I(XBEGIN + (K-1)H)$$

On input $Y(I,1)$ are given initial values for $1 \leq I \leq N$

G is an N x M array given on input for all I, K

$$G(I,K) = g_I(XBEGIN+(K-1)H)$$

A is an N x N x M array given on input for all I, J, K

$$A(I,J,K) = a_{IJ}(XBEGIN+(K-1)H)$$

Solving the problem is defined in an unusual way. We want solution values on the given mesh as accurate as it is possible to get them. In particular, it is the true, or global, error we wish to control rather than the local error that is usually controlled. It is important that we ask the user to interact with the solution process only if it is unavoidable. It may be presumed that the number of equations and the number of mesh points is not so large as to cause storage difficulties and that any moderately efficient solution procedure will result in acceptable computing times.

3. A Sketch of the Algorithm

We shall present an algorithm and a code for realizing it. The code listed in this document is written in FLECS [3] which is a FORTRAN preprocessor. The output of the preprocessor is a FORTRAN subroutine which can be run in SPEAKEASY. However, the reason for using FLECS is that the control structures available make the workings of the algorithms transparent when coded. This is appropriate for the explanations provided by this document and is also very convenient for translation to other languages. We have also used the Basic Linear Algebra Subprograms [4] which are standard FORTRAN modules for performing basic operations such as adding two vectors. These are operations facilitated by SPEAKEASY so that not only do they clarify the coding, they suggest to us that it would be easy to write the subroutine in the SPEAKEASY language itself were

this to be deemed desirable.

FLECS makes describing the code quite easy. Leaving aside the call list, comments, dimension statements, and data statements, the code itself is merely

```
INITIALIZE-COEFFICIENTS-AND-DIVISORS
DO-SWEEP0
IF(MUSED.NE.1)DO-UP-TO-FOUR-MORE-SWEEPS
PREPARE-TO-RETURN
RETURN
```

The statements here involve procedures in FLECS which may be thought of as blocks of code or subroutines. First we initialize the coefficients defining the formulas. They are set via assignment statements because this method makes the indices very clear. Also, the coefficients are all integers or ratios of integers, the largest of which is 65112, so that there should be no trouble getting them into any medium sized computer accurately. Sweep 0 does some initialization and determines how much of the integration is possible with the given step size. MUSED is the number of mesh points actually used in the integration. If there is more than one mesh point, we successively sweep through the mesh generating more and more accurate solution values. When this process yields no more improvement, we gather together information the user needs: We tell him the interval on which the problem was successfully solved and how accurately, and what to do to get a more accurate solution. We also tell him what interval remains to be solved and what action he must take to solve it. Finally we do a FORTRAN RETURN from the subroutine.

In succeeding sections we shall describe the algorithms and their development more fully. The FLECS code is heavily commented and we shall often refer to it to connect up the theory and practice.

4. Choice of a Basic Method

The constraints imposed by the structure of the SPEAKEASY language greatly limit our options for a basic method. The fundamental difficulty that the equation is defined only at discrete points is one of some general importance because it occurs when any function in a differential equation arises from experimental data. There are two distinct approaches possible: One is to fit the data so as to generate functions which we can evaluate anywhere and then solve the approximating problem as accurately as we like using any one of the pieces of mathematical software generally available. The other is to employ a numerical method which uses only the discrete data available. Both approaches require us to address the problem of the limited accuracy inherent in the discretization. It is not clear how to assess this in the first approach. In the second approach, because our data is assumed exact, the discretization error is exactly that which we are accustomed to dealing with. The only difference is that we cannot adjust the discretization to do anything about it. Thus although the first approach is the easier mechanically, it appears to be far more difficult conceptually. Also, because we seek to do a much better job in our special circumstances than a general purpose code would do, the effort of developing a new integration scheme cannot really be avoided.

Although the user could specify in SPEAKEASY a set of points on which he wishes an approximate solution, in general he is unlikely to have available the information and knowledge as to how to do this. Furthermore it is a basic principle that the solution ought to be as simple as possible. We are led to computing a solution at equally spaced points. A natural development here is to compute the solution as far as possible on such a mesh and to tell the user that he must use a finer mesh on the remainder

of the interval. This actually accomplishes a crude variation in the mesh if it is absolutely necessary.

The standard initial value techniques are essentially excluded by the circumstances. Runge-Kutta methods require evaluations at off-mesh points so cannot be used. Methods involving memories, such as the Adams methods, have a starting problem. Starting is usually accomplished by either varying the mesh initially or by a Runge-Kutta method, both of which are excluded. Extrapolation methods in their usual form involve a refinement of the mesh, which is excluded, and regarding the given mesh as the finest in a set of refinements means that solution values are not computed at all the mesh points.

In context it is important to get all possible accuracy on the given mesh. We cannot vary the mesh so varying the formula is our only option. The only way we really understand for doing this amounts to using procedures of successively higher orders. An appealing procedure is that of difference correction which successively raises the accuracy of solution on a fixed mesh by applying higher order methods. We settled on a family of block one-step methods [5] which can be viewed as variants of a number of the procedures mentioned. The solution is advanced by a block of new values at a time, e.g. if we use the notation $\underline{y}_n \doteq \underline{y}(x_n)$, one might generate $\underline{y}_1, \underline{y}_2$ and then $\underline{y}_3, \underline{y}_4$, etc. The process can be viewed as a Runge-Kutta procedure which happens to do all its evaluations at mesh points and for which the intermediate computations represent solution approximations of the same order as the final value at the end of the "step." The scheme we employ uses the highest order possible which, as it turns out, can be viewed as arising in the same way as the Adams formulas but differs by simultaneously getting a block of new

solution values rather than remembering all but one from previous computations. Applying these block methods so that the order is increased on successive sweeps of solution is in essence a difference correction procedure. It is our intention to develop the code further to deal with stiff problems, so it is very convenient that these formulas are also suitable for this task. The formulas have the form

$$y_{n+i} = y_n + h \sum_{j=0}^b \alpha_{ij} (a(x_{n+j})y_{n+j} + g(x_{n+j})) \quad i = 1, \dots, b$$

when a block of b new values is generated at a time. The order which can be achieved by the Newton-Cotes family [5, Appendix C]

block length b	1	2	3	4	5
order of convergence	2	4	6	8	10

and all these formulas are A-stable. It is not likely that very fine meshes will be used in SPEAKEASY because of the storage implications. For this reason very high orders are not likely to be used and our experiments showed that the order 10 results were never used in what seemed to be normal computation. And so we limited our formulas to order eight.

5. Necessary Conditions on the Step Size

Because the step size is given to the code without knowledge of the behavior of the solution, it is quite possible that it is so large as to cause the computations to be unstable (to "blow up") or to cause our judgements based on asymptotic behavior to fail because they are not applicable. This situation does not arise so seriously in more typical computation because codes which vary their step size do so to maintain stability and help assure the validity of the asymptotic assumptions.

Suppose that at x_j there is an error of amount $\underline{\delta}_j$ in the solution. If we were to continue the integration with no further error, we would compute $\underline{u}(x)$ from

$$\begin{aligned}\underline{u}' &= a(x)\underline{u} + \underline{g}(x) \\ \underline{u}(x_j) &= \underline{y}(x_j) + \underline{\delta}_j .\end{aligned}$$

The difference between $\underline{u}$ and the true solution, $\underline{e}(x) = \underline{u}(x) - \underline{y}(x)$, satisfies

$$\underline{e}' = a(x)\underline{e} , \quad \underline{e}(x_j) = \underline{\delta}_j .$$

Thus, expanding $\underline{e}(x_j+h)$ in terms of the step size h , we see that

$$\begin{aligned}\underline{e}(x_j+h) &= \underline{e}(x_j) + h\underline{e}'(x_j) + O(h^2) \\ &= \underline{\delta}_j + ha(x_j)\underline{\delta}_j + O(h^2) .\end{aligned}$$

Clearly the propagation of the error is unstable unless $ha(x_j)$ is small in some sense. A minimal requirement would appear to be something like

$$(*) \quad \|ha(x_j)\| < 1$$

for some matrix norm. It can happen that the matrix a implies that the error $\underline{\delta}_j$ will be strongly damped but that $\|a\|$ is very large. For such problems, termed stiff problems, the requirement $(*)$ can be unrealistically severe. The code described herein is aimed at non-stiff problems for which $\|a\|$ is of moderate size and we shall consider stiff problems at another time.

We have already stated the form of the formulas which are to be the basis of our computation:

$$\underline{y}_{n+1} = \underline{y}_n + h \sum_{j=0}^b \alpha_{ij} (a(x_{n+j})\underline{y}_{n+j} + \underline{g}(x_{n+j})) \quad i = 1, \dots, b$$

It is our intention to solve for the $\underline{y}_{n+i}$ by simple iteration, i.e. given $\underline{y}_{n+i,0}$ we iterate by

$$\underline{y}_{n+i,m+1} = \underline{y}_n + h \sum_{j=0}^b \alpha_{ij} (a(x_{n+j}) \underline{y}_{n+j,m} + \underline{g}(x_{n+j})) \quad i = 1, \dots, b.$$

If we let the iteration error be $\underline{e}_{i,m} = \underline{y}_{n+i,m} - \underline{y}_{n+i}$, we see that

$$\underline{e}_{i,m+1} = h \sum_{j=0}^b \alpha_{ij} a(x_{n+j}) \underline{e}_{j,m} \quad i = 1, \dots, b$$

Now let us define

$$|a| = \max_{x_i} \|a(x_i)\|$$

$$|e_m| = \max_{1 \leq i \leq b} \|e_{i,m}\|$$

$$\alpha = \max_{1 \leq i \leq b} \sum_{j=0}^n |\alpha_{ij}|$$

for suitable vector and matrix norms. Then it is easy to see that

$$|e_{m+1}| \leq |e_m| |h| \alpha |a|.$$

The error of iteration will tend to zero and the method will be well defined if $|h| \alpha |a| < 1$. This is more stringent than (*).

For the methods considered $\alpha < 5.72$ so that if we require

$$|h| < \frac{1}{10 |a|},$$

we are assured that the numerical methods are well defined and that the error will not propagate too fast.

This test will be applied to see how far we might reasonably try to

integrate and to deduce how many equally spaced mesh points appear necessary for the integration of the remainder of the interval. A matrix norm which is cheap to compute is

$$\|a(x)\| = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{ij}(x)|$$

and we shall use it in the test.

The procedure DO-SWEEPO does some initialization first -- it computes the step size and defines the initial solution to be a constant over the interval. Then it runs through the interval testing if $|h| < 1/10|a|$. The integration will proceed from XBEGIN as far towards XEND as this condition is satisfied, namely XQUIT. The formulas require at least 8 points so if the condition is not satisfied for this many points we cannot integrate at all and must set XQUIT=XBEGIN. MUSED is the number of points actually used, that is, the number of mesh points in the interval from XBEGIN to XQUIT. If $XQUIT \neq XEND$, we must run through the remainder of the interval computing $10|a|$ so that we can tell the user how fine a mesh he must use if he is to integrate from XQUIT to XEND. The actual computation of the necessary number of mesh points is done in the procedure PREPARE-TO-RETURN.

6. Varying the Order

The basic idea for computing solutions as accurately as possible on the given mesh and for estimating the global error is to compute independently solutions of increasingly high order and compare them. We start off with a solution of order 0 (a constant solution) from the initialization in sweep 0. Then in sweep L we carry out the integration using a formula of order H^{2L} based on a block involving $\max(2, 2L-1)$ points.

We compare the solution at each mesh point to that of the previous sweep, which is of order $H^{2(L-1)}$. Our hypothesis is that the higher order solution is more accurate until we find evidence that this is not true. Certainly for "small" step sizes the higher order result is much more accurate and provides an excellent estimate of the global error E_{L-1} of the result of sweep(L-1). We chose to increase the order of the formulas by two at each sweep to improve the quality of the estimate as well as to get to high order results quickly. In sweep L we form E_{L-1} . If $L = 1$, we automatically raise the order and go to $L = 2$. If $L > 1$, we compare E_{L-1} to E_{L-2} . If $E_{L-1} \leq E_{L-2}$, we conclude that raising the order is improving the solution and we raise the order unless we have already reached the maximum of $L = 4$. If $E_{L-1} > E_{L-2}$ or if $L = 4$, we stop raising the order and accept the result of order $H^{2(L-2)}$ or H^6 as our answer, respectively. As we have previously noted, for practical meshes there seemed to be no point in doing more than four sweeps.

The procedure DO-UP-TO-FOUR-MORE-SWEEPS carries out the algorithm just outlined. It calls upon a procedure to generate a new solution of order H^{2L} . Then it calls upon a procedure to estimate the global error of the result of the preceding sweep. This is done by comparison as we have said but the details will be discussed in the next section. Finally it is decided whether or not to raise the order by doing another sweep. The basic linear algebra subprogram SCOPY is used to swap solution arrays so as to retain only those needed for the tests in order to hold down storage used.

Generating a solution is done in the procedure GENERATE-A-SOLUTION and is straightforward with one exception. We are using block methods so that there is a problem at the end of the integration. What we do is to

discard some solution values from the next to last block so that we can end up at XQUIT. We might remark that this is the analog of the starting problem for methods involving memory but we have a far simpler solution for this special class of formulas. The computation of a block of new solution values is done in the procedure COMPUTE-A-BLOCK. Recall that this is done by iteration:

$$\underline{y}_{n+i,m+1} = \underline{y}_n + h \sum_{j=0}^b \alpha_{ij} (a(x_{n+j}) \underline{y}_{n+j,m} + \underline{g}(x_{n+j})) \quad i = 1, \dots, b$$

The quantity

$$a(x_{n+j}) \underline{y}_{n+j,m} + \underline{g}(x_{n+j})$$

is, from the differential equation, the natural thing to call the approximate derivative

$$\underline{y}'_{n+j,m}$$

so we have a procedure COMPUTE-DERIVATIVES which evaluates all these derivatives for a block. All COMPUTE-A-BLOCK does is carry out four iterations to get the new solution values. The initial guesses $\underline{y}_{n+i,0}$ are taken from the previous sweep so have an accuracy of order $H^{2(L-1)}$. The formulas being used in sweep L have a global error of order H^{2L} which corresponds to a local error of order H^{2L+1} . It is easy to see that each iteration increases the local accuracy by one order so that we need three or more iterations with these starting values to get to the correct order. It would be cheaper to use a predictor instead of the last global solution but it would probably reduce the stability of the computation greatly [6] unless we did extra iterations to stabilize it. Anyway, iterations are relatively cheap in this context and we preferred the simpler approach.

It is worth remark that the stability properties of these formulas as we implement them are unusual. There is no propagated error from a predictor formula which must be damped out by iteration. We did choose to do four iterations to wash out the effects of the $y_{n+1,0}$ and make the stability properties essentially those of the formulas employed in the usual manner. Four iterations should lead to very large stability regions (not scaled for work which is not pertinent here) but we have not yet plotted them. The stability properties of the formulas as implemented could stand some more theoretical investigation. If we changed COMPUTE-A-BLOCK so as to solve the (linear) equations for the new values $y_{n+1,i}$ exactly, the resulting formulas would be A-stable. This will be the key to solving stiff problems but there are other serious questions we have not yet answered in this connection.

7. Error Measurement

To decide whether to raise the order and, at the end of the computation, to assess the merit of the solution we must somehow measure the error. The measurement must be simple enough that we can convey easily to the user the quality of the solution and it must be realistic and safe. A pure absolute error measure is not realistic because it does not reflect the scale of the solution. A pure relative error takes scale into account but is unsafe at, or very near, a zero of the solution. We have chosen to measure the error in an absolute sense if the solution is less than one in magnitude and in a relative sense if the solution is greater than one in magnitude. For our purposes we are concerned with the error throughout the range of integration so it is the worst error we compute. To keep matters simple we also consider only the worst error in any equation.

The error in the $O(H^{2(L-1)})$ solution of equation I is measured by comparing the solution to that of the $O(H^{2L})$ solution. Thus

$$E_{L-1} = \max_{K \text{ in interval eqn I}} \max \frac{|(O(H^{2L}) \text{ sol'n} - O(H^{2(L-1)}) \text{ sol'n})_{\text{eqn I}}|}{\max\{1, |O(H^{2L}) \text{ sol'n eqn I}|\}}$$

which is computed in the procedure COMPUTE-GLOBAL-ERROR.

If the user of the code is dissatisfied with the estimated maximum global error, he will have to solve the problem again with a smaller step size. But how much smaller? We choose to return to him a factor R such that if he uses R times as many mesh points in a new computation, he should get about one order of magnitude more accuracy.

Suppose that the accepted solution is of order 2L so that the error is (asymptotically) proportional to H^{2L} . We wish to increase the number of mesh points by a factor R, which is equivalent to reducing H by a factor of $1/R$, so that the error is reduced by a factor of 0.1. Thus

$$R = \frac{1}{(0.1)^{\frac{1}{2L}}} = 10^{\frac{1}{2L}}.$$

Unfortunately it is possible that the solution accepted be of order 0, meaning that the step size is not small enough for asymptotic results to be relevant. In such a case we have no guide for choosing a factor R. We arbitrarily take $R = 10$ in such a case. Notice that if the order were two, the factor would be about 2 so this is not way out of line if we should go to order two in the next computation by virtue of having attained a small enough step size that asymptotic results are meaningful, yet it is large enough that we can rapidly get into this range. The computation of R is done in the procedure PREPARE-TO-RETURN.

The R selected in this way may be quite conservative because it is based on the order used in the given computation. As a general rule, if the step size is small enough, a high order method will be more accurate than a lower order one. Thus as the mesh is refined the code will tend to go to higher orders. If an R is selected at one order and in a subsequent computation the code goes to a higher order, we expect that the error will be reduced by more than a factor of 10.

8. Some Examples

Our first example will clarify the role of the preliminary tests on the mesh made in sweep 0 of the code. Consider the problem

$$\underline{y}' = \begin{pmatrix} -10 & 0 \\ 0 & -20 \end{pmatrix} \underline{y} + \begin{pmatrix} e^{-10x} \\ xe^{-20x} \end{pmatrix},$$

$$\underline{y}(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ integrate from 0 to 10.}$$

This is a simple example of a stiff problem, meaning here that an unrealistically small step size would be necessary to solve the problem with a code like ours -- although it will do so if you are willing to use this step size. When the code was called with eleven mesh points, it returned with the information that XQUIT=XBEGIN=0 and MUSED=1, meaning that it did not advance the solution at all, and with MMORE=2001, meaning that at least 2001 mesh points would be necessary to integrate this problem. A "large" value of MMORE is the way that stiffness will be reported. However, there are other conditions which could result in large values returned in this parameter. It is presumed that the solution components are smooth. It is not possible for the code to be presented a problem in

SPEAKEASY which exhibits a discontinuity or singularity. This is because the code has only the finite values given at mesh points to work with. It cannot distinguish between problems which are extremely smooth between mesh points and those becoming infinite there; it presumes they are smooth. It may be that the problem changes abruptly enough that this is reflected in the discrete data so that the code will return a large value of MMORE. A much sharper indication of a lack of smoothness is the parameter R and we shall discuss it in a moment.

The next example might be termed a normal one. We have

$$\underline{y}' = \begin{pmatrix} 0 & -2x \\ 2x & 0 \end{pmatrix} \underline{y} + \begin{pmatrix} 2x \\ 0 \end{pmatrix},$$

$$\underline{y}(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ integrate from 0 to 2}$$

for which the solution is

$$\underline{y}(x) = \begin{pmatrix} \cos(x^2) \\ 1 + \sin(x^2) \end{pmatrix}.$$

As x increases the solution components oscillate increasingly rapidly so that with a constant mesh spacing we should expect the solution to become increasingly difficult in x. When the code was called with 41 mesh points it returned with the result that it had successfully integrated from 0 to XQUIT=1 using 21 points of the original mesh. The worst global error was estimated to be 1.0×10^{-5} in the measure discussed in the preceding section. When we computed the true global error by comparison with the known solution we found that the worst error was 1.0×10^{-5} . To reduce the error by a factor of 10 the code estimated that the number of mesh points would need to be increased by a factor $R = 1.78$. The code also

returns information about the solution of the remainder of the interval. It said that at least 41 points ought to be used. When we solved the remaining interval of 1 to 2 with 42 points, the computation succeeded. The estimated global error was 8.5×10^{-6} and the true global error was 1.0×10^{-5} .

The third problem involves only one equation and integrates in the direction of decreasing x .

$$y' = \frac{-8(x-1)^7}{1+(x-1)^8} y ,$$

$$y(2) = 5 \quad \text{integrate from 2 to 0 .}$$

The solution

$$y(x) = \frac{10}{1+(x-1)^8}$$

has a fairly sharp peak at $x = 1$. We first called the code with too few mesh points and it promptly stated that at least 81 mesh points were needed. We tried 101, which was successful. The solution was computed over the entire interval with an estimated error of 4.5×10^{-6} . The true error was 4.4×10^{-6} . The code said that if the number of mesh points were increased by a factor of 1.78, we should reduce the error by a factor of about 10. So, we solved the problem again with 180 mesh points. This resulted in a solution with an estimated error of 6.5×10^{-7} and a true error of 6.5×10^{-7} . In this case the prediction about the error reduction worked very well. It should if the asymptotic results are valid. If the estimated error were not improved by anything like a factor of 10, this would be good evidence that the asymptotic approximations are invalid because the problem is not smooth, e.g. a discontinuity or singularity

is present, or that we have reached the limits imposed by the number of digits available with our computer.

References

1. S. Cohen and S. C. Pieper, The SPEAKEASY-3 Reference Manual, Rept. ANL-8000 Rev. 1, Math. and Computers, Argonne National Laboratory, Argonne, Illinois, 1976.
2. L. F. Shampine and H. A. Watts, Global error estimation for ordinary differential equations, ACM Trans. Math. Software, 2(1976) 172-186. Also see Algorithm 504 GERK, pp. 200-203 of the same reference.
3. T. Beyer, FLECS -- Fortran language with extended control constructs. User's manual. University of Oregon Computing Center, Eugene, Oregon, 1974.
4. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, Basic linear algebra subprograms for FORTRAN usage, SAND77-0898J, Sandia Laboratories, Albuquerque, NM, 1977.
5. H. A. Watts, A-stable block implicit one-step methods, SC-RR-71 0296, Sandia Laboratories, Albuquerque, NM, 1971. Also see H. A. Watts and L. F. Shampine, A-stable block implicit one-step methods, BIT, 12(1972) 252-266.
6. L. F. Shampine and H. A. Watts, Block implicit one-step methods, Math. Comp., 23(1969) 730-740.

```

00001      SUBROUTINE SPKODE(N,A,Y,G,XBEGIN,XEND,M,XQUIT,MUSED,ERROR,R,MORE,
00002      1      Y1,Y2,YP,TEMP)
00003  C
00004  C
00005  C      THIS CODE SOLVES A SYSTEM OF N LINEAR DIFFERENTIAL
00006  C      EQUATIONS OF THE FORM
00007  C
00008  C       $dy(1,*)/dx = A(1,1,*)y(1,*) + \dots + A(1,N,*)y(N,*) + G(1,*)$ 
00009  C       $\vdots$ 
00010  C       $\vdots$ 
00011  C       $\vdots$ 
00012  C       $dy(N,*)/dx = A(N,1,*)y(1,*) + \dots + A(N,N,*)y(N,*) + G(N,*)$ 
00013  C
00014  C      FROM XBEGIN TO XEND ON A MESH OF M EQUALLY SPACED POINTS.
00015  C      IT IS NECESSARY THAT M.GE.7. THE SPACING IS GIVEN BY
00016  C       $H = (XEND - XBEGIN)/(M-1)$  SO MESH POINT K REPRESENTS
00017  C       $X = XBEGIN + (K-1)*H$ .
00018  C
00019  C      THE SOLUTION ARRAY  $Y(*,*)$  IS N BY M WHERE  $Y(I,K)$  REPRESENTS
00020  C      THE SOLUTION OF EQUATION I AT MESH POINT K. ON INPUT THE
00021  C      INITIAL VALUE FOR EACH SOLUTION COMPONENT AT XBEGIN MUST BE
00022  C      GIVEN, THAT IS,  $Y(I,1)$  MUST BE SPECIFIED FOR EACH I,
00023  C       $I = 1, \dots, N$ .
00024  C
00025  C      ON INPUT THE N BY N BY M ARRAY  $A(I,J,K)$  AND THE N BY M
00026  C      ARRAY  $G(I,K)$  MUST BE GIVEN FOR  $I = 1, \dots, N$ ,  $J = 1, \dots, N$ ,
00027  C       $K = 1, \dots, M$ .
00028  C
00029  C      THE CODE WILL INTEGRATE FROM XBEGIN AS FAR AS POSSIBLE
00030  C      TOWARDS XEND AND WILL RETURN AS ACCURATE A SOLUTION AS
00031  C      POSSIBLE IN  $Y(*,*)$ .
00032  C
00033  C      ON OUTPUT THE PROBLEM HAS BEEN SOLVED FROM XBEGIN TO XQUIT
00034  C      USING MUSED OF THE ORIGINAL MESH POINTS. THE WORST ERROR
00035  C      MADE IN THE SOLUTION IS ESTIMATED TO BE ERROR. IT IS AN
00036  C      ABSOLUTE ERROR FOR SOLUTION VALUES LESS THAN ONE IN
00037  C      MAGNITUDE AND RELATIVE ERROR OTHERWISE.
00038  C      TO REDUCE THIS ERROR TO ABOUT  $ERROR/10**P$ , SOLVE THE PROBLEM
00039  C      AGAIN ON THIS INTERVAL USING  $MUSED*R**P$  MESH POINTS.
00040  C
00041  C      IF MUSED.LT.M, IT WAS IMPOSSIBLE TO INTEGRATE FROM XQUIT TO
00042  C      XEND BECAUSE THE MESH WAS NOT FINE ENOUGH. AT LEAST MORE
00043  C      MESH POINTS WILL BE NEEDED TO SOLVE THE PROBLEM ON THIS
00044  C      INTERVAL. IF MORE SEEMS UNREASONABLY LARGE, THE COEFFICIENTS
00045  C      DEFINING THE EQUATION MAY NOT BE SMOOTH OR THE PROBLEM MAY BE
00046  C      STIFF. THIS CODE IS NOT INTENDED FOR EITHER SITUATION.
00047  C
00048  C
00049  C      SUMMARY
00050  C
00051  C      THE CALLING PROGRAM MUST PROVIDE STORAGE FOR THE ARRAYS
00052  C       $A(N,N,M)$ ,  $Y(N,M)$ ,  $G(N,M)$ ,  $Y1(N,M)$ ,  $Y2(N,M)$ ,  $YP(N,9)$ ,  $TEMP(M)$ .
00053  C
00054  C      THE USER DEFINES THE PROBLEM WITH THE INPUT VARIABLES N,A,Y,G,
00055  C      XBEGIN,XEND AS DESCRIBED, AND INDICATES HOW MANY MESH POINTS,
00056  C      M.GE.7, ARE TO BE USED IN THE SOLUTION.
00057  C
00058  C      THE OUTPUT VARIABLES Y,XQUIT,MUSED,ERROR,R,MORE ALONG WITH THE
00059  C      UNCHANGED VALUES OF XBEGIN, XEND REPORT THE SOLUTION AND
00060  C      SUGGEST FURTHER ACTION BY THE USER.
00061  C

```

```

00062 C    THE ARRAYS Y1,Y2,YP,TEMP ARE WORKING STORAGE AND THEIR CONTENT
00063 C    WILL BE OF NO INTEREST TO THE USER.
00064 C
00065 C
00066 C    LOGICAL FNDLIM,DONE
00067 C    INTEGER ORDER
00068 C    DIMENSION A(N,N,1),G(N,1),Y(N,1),Y1(N,1),Y2(N,1)
00069 C    DIMENSION TEMP(N),YP(N,7)
00070 C    DIMENSION COEFF(7,6,4),DIVSOR(6,4)
00071 C    DATA ZERO,ONE,TEN/0.E0,1.E0,1.E1/
00072 C
00073 C
00074 C
00075 C    INITIALIZE-COEFFICIENTS-AND-DIVISORS
00076 C    DO-SWEEP0
00077 C    IF(MUSED.NE.1) DO-UP-TO-FOUR-MORE-SWEEPS
00078 C    PREPARE-TO-RETURN
00079 C    RETURN

```

```

-----
00080 C    TO DO-SWEEP0
00081 C
00082 C    . IN THIS SWEEP SOME INITIALIZATION IS DONE AND IT IS
00083 C    . DETERMINED HOW MUCH OF THE INTERVAL CAN BE INTEGRATED
00084 C    . WITH THE STEP SIZE GIVEN.
00085 C
00086 C    . H = (XEND-XBEGIN)/FLOAT(M-1)
00087 C
00088 C    . DEFINE THE 0TH ORDER SOLUTION AS M COPIES OF
00089 C    . THE INITIAL CONDITIONS.
00090 C
00091 C    . DO(K=2,M)
00092 C    . . CALL SCOPY(N,Y,1,Y(1,K),1)
00093 C    . ...FIN
00094 C
00095 C    . FIND LIM, THE FIRST K FOR WHICH THE
00096 C    . (KTH MAX COLUMN SUM MATRIX NORM)*10*H .GT. 1.
00097 C    . DEFAULT CASE, LIM = M+1. COMPUTE ANRM, THE LARGEST MATRIX
00098 C    . NORM IN THE WHOLE INTERVAL.
00099 C
00100 C    . FNDLIM = .FALSE.
00101 C    . LIM = M+1
00102 C    . ANRMAX = ZERO
00103 C    . DO(K=2,M)
00104 C
00105 C    . . COMPUTE NORM OF A AT MESH POINT K.
00106 C
00107 C    . . ANORM = ZERO
00108 C    . . DO(J=1,N)
00109 C    . . . ANORM = AMAX1(ANORM,SASUM(N,A(1,J,K),1))
00110 C    . . ...FIN
00111 C
00112 C    . . COMPUTE MAXIMUM OF ALL MATRIX NORMS IN THE INTERVAL.
00113 C
00114 C    . . ANRMAX = AMAX1(ANRMAX,ANORM)
00115 C
00116 C    . . DETERMINE LIM.
00117 C
00118 C    . . APROD = TEN*H*ANORM
00119 C    . . IF(APROD.GT.ONE.AND.(.NOT.FNDLIM))
00120 C    . . . LIM = K
00121 C    . . . FNDLIM = .TRUE.
00122 C    . . ...FIN

```

```

00123 . ...FIN
00124 C .
00125 C . WITH THIS STEP SIZE THE INTEGRATION CAN BE DONE ONLY AS
00126 C . FAR AS XQUIT USING MUSED POINTS.
00127 C .
00128 . WHEN(LIM.GT.7)
00129 . . XQUIT = XBEGIN + FLOAT(LIM-2)*H
00130 . . MUSED = LIM - 1
00131 . . IF(MUSED.EQ.M) XQUIT = XEND
00132 . ...FIN
00133 . ELSE
00134 . . XQUIT = XBEGIN
00135 . . MUSED = 1
00136 . ...FIN
00137 ...FIN

```

```

-----
00138 TO INITIALIZE-COEFFICIENTS-AND-DIVISORS
00139 C .
00140 C . SUBSCRIPT 1 OF COEFF CORRESPONDS TO THE ITH TERM IN THE FORMULA.
00141 C . SUBSCRIPT 2 OF COEFF CORRESPONDS TO THE JTH FORMULA.
00142 C . SUBSCRIPT 3 OF COEFF CORRESPONDS TO THE KTH METHOD.
00143 C .
00144 C . SUBSCRIPT 1 OF DIVSOR CORRESPONDS TO THE ITH FORMULA.
00145 C . SUBSCRIPT 2 OF DIVSOR CORRESPONDS TO THE JTH METHOD.
00146 C .
00147 C . INITIALIZE ARRAYS TO ALL ZEROES.
00148 C .
00149 . CALL SCOPY(168,ZERO,0,COEFF,1)
00150 . CALL SCOPY(24,ZERO,0,DIVSOR,1)
00151 C .
00152 C . METHOD 1.
00153 C .
00154 . COEFF(1,1,1) = 1.E0
00155 . COEFF(2,1,1) = 1.E0
00156 . DIVSOR(1,1) = 1.E0/2.E0
00157 C .
00158 C . METHOD 2.
00159 C .
00160 . COEFF(1,1,2) = 5.E0
00161 . COEFF(2,1,2) = 8.E0
00162 . COEFF(3,1,2) = -1.E0
00163 . DIVSOR(1,2) = 1.E0/12.E0
00164 . COEFF(1,2,2) = 1.E0
00165 . COEFF(2,2,2) = 4.E0
00166 . COEFF(3,2,2) = 1.E0
00167 . DIVSOR(2,2) = 1.E0/3.E0
00168 C .
00169 C . METHOD 3.
00170 C .
00171 . COEFF(1,1,3) = 251.E0
00172 . COEFF(2,1,3) = 646.E0
00173 . COEFF(3,1,3) = -264.E0
00174 . COEFF(4,1,3) = 106.E0
00175 . COEFF(5,1,3) = -19.E0
00176 . DIVSOR(1,3) = 1.E0/720.E0
00177 . COEFF(1,2,3) = 29.E0
00178 . COEFF(2,2,3) = 124.E0
00179 . COEFF(3,2,3) = 24.E0
00180 . COEFF(4,2,3) = 4.E0
00181 . COEFF(5,2,3) = -1.E0
00182 . DIVSOR(2,3) = 1.E0/90.E0
00183 . COEFF(1,3,3) = 9.E0

```

```

00184 . COEFF(2,3,3) = 34.E0
00185 . COEFF(3,3,3) = 24.E0
00186 . COEFF(4,3,3) = 14.E0
00187 . COEFF(5,3,3) = -1.E0
00188 . DIVSOR(3,3) = 3.E0/80.E0
00189 . COEFF(1,4,3) = 7.E0
00190 . COEFF(2,4,3) = 32.E0
00191 . COEFF(3,4,3) = 12.E0
00192 . COEFF(4,4,3) = 32.E0
00193 . COEFF(5,4,3) = 7.E0
00194 . DIVSOR(4,3) = 2.E0/45.E0
00195 C .
00196 C . METHOD 4.
00197 C .
00198 . COEFF(1,1,4) = 19087.E0
00199 . COEFF(2,1,4) = 65112.E0
00200 . COEFF(3,1,4) = -46461.E0
00201 . COEFF(4,1,4) = 37504.E0
00202 . COEFF(5,1,4) = -20211.E0
00203 . COEFF(6,1,4) = 6312.E0
00204 . COEFF(7,1,4) = -836.E0
00205 . DIVSOR(1,4) = 1.E0/60480.E0
00206 . COEFF(1,2,4) = 1139.E0
00207 . COEFF(2,2,4) = 5640.E0
00208 . COEFF(3,2,4) = 33.E0
00209 . COEFF(4,2,4) = 1328.E0
00210 . COEFF(5,2,4) = -807.E0
00211 . COEFF(6,2,4) = 264.E0
00212 . COEFF(7,2,4) = -37.E0
00213 . DIVSOR(2,4) = 1.E0/3780.E0
00214 . COEFF(1,3,4) = 685.E0
00215 . COEFF(2,3,4) = 3240.E0
00216 . COEFF(3,3,4) = 1161.E0
00217 . COEFF(4,3,4) = 2176.E0
00218 . COEFF(5,3,4) = -729.E0
00219 . COEFF(6,3,4) = 216.E0
00220 . COEFF(7,3,4) = -29.E0
00221 . DIVSOR(3,4) = 1.E0/2240.E0
00222 . COEFF(1,4,4) = 143.E0
00223 . COEFF(2,4,4) = 696.E0
00224 . COEFF(3,4,4) = 192.E0
00225 . COEFF(4,4,4) = 752.E0
00226 . COEFF(5,4,4) = 87.E0
00227 . COEFF(6,4,4) = 24.E0
00228 . COEFF(7,4,4) = -4.E0
00229 . DIVSOR(4,4) = 2.E0/945.E0
00230 . COEFF(1,5,4) = 743.E0
00231 . COEFF(2,5,4) = 3480.E0
00232 . COEFF(3,5,4) = 1275.E0
00233 . COEFF(4,5,4) = 3200.E0
00234 . COEFF(5,5,4) = 2325.E0
00235 . COEFF(6,5,4) = 1128.E0
00236 . COEFF(7,5,4) = -55.E0
00237 . DIVSOR(5,4) = 5.E0/12096.E0
00238 . COEFF(1,6,4) = 41.E0
00239 . COEFF(2,6,4) = 216.E0
00240 . COEFF(3,6,4) = 27.E0
00241 . COEFF(4,6,4) = 272.E0
00242 . COEFF(5,6,4) = 27.E0
00243 . COEFF(6,6,4) = 216.E0
00244 . COEFF(7,6,4) = 41.E0
00245 . DIVSOR(6,4) = 1.E0/140.E0
00246 . . . FIN

```

```

-----
00247      TO DO-UP-TO-FOUR-MORE-SWEEPS
00248      .   DONE = .FALSE.
00249      .   ELM2 = ZERO
00250 C     .
00251 C     .   COPY SOLUTION IN Y TO Y1 AND Y2.
00252 C     .
00253      .   DO (K=1,M)
00254      .       .   CALL SCOPY(N,Y(1,K),1,Y1(1,K),1)
00255      .       .   CALL SCOPY(N,Y(1,K),1,Y2(1,K),1)
00256      .       ...FIN
00257 C     .
00258      .   DO(L=1,4)
00259      .       .   UNLESS(DONE)
00260 C     .
00261 C     .       .   GENERATE A SOLUTION IN Y2(*,*) OF GLOBAL ERROR OF ORDER
00262 C     .       .       H**(2L) USING A BLOCK FORMULA INVOLVING MAX(2,2L-1)
00263 C     .       .       POINTS.
00264 C     .
00265      .       .   GENERATE-A-SOLUTION
00266 C     .
00267 C     .       .   Y CONTAINS THE SOLUTION OF ORDER H**(2(L-2)), THE
00268 C     .       .       CURRENT ACCEPTED SOLUTION. ELM2 IS ITS ESTIMATED ERROR.
00269 C     .
00270 C     .       .   ELM1, THE GLOBAL ERROR OF THE SOLUTION IN Y1(*,*) OF
00271 C     .       .       ORDER H**(2(L-1)), IS ESTIMATED BY COMPARISON TO THE
00272 C     .       .       SOLUTION IN Y2(*,*) OF ORDER H**(2L).
00273 C     .
00274      .       .   COMPUTE-GLOBAL-ERROR
00275 C     .
00276 C     .       .   IF L=1 OR ELM1.LE.ELM2, ACCEPT THE ORDER H**(2*(L-1))
00277 C     .       .       SOLUTION AND LOOP TO TRY A HIGHER ORDER. OTHERWISE, SET
00278 C     .       .       DONE TO .TRUE. TO FORCE EXIT FROM THE LOOP.
00279 C     .
00280      .       .   WHEN(L.EQ.1.OR.ELM1.LE.ELM2)
00281      .       .       .   DO(K=2,M)
00282      .       .       .       .   CALL SCOPY(N,Y1(1,K),1,Y(1,K),1)
00283      .       .       .       .   CALL SCOPY(N,Y2(1,K),1,Y1(1,K),1)
00284      .       .       .       ...FIN
00285      .       .       .   ELM2 = ELM1
00286      .       .       .   ...FIN
00287      .       .       ELSE
00288      .       .       .   ORDER = 2*(L-2)
00289      .       .       .   DONE = .TRUE.
00290      .       .       .   ...FIN
00291      .       .       ...FIN
00292      .       .   ...FIN
00293      .   UNLESS(DONE) ORDER = 6
00294      .   ...FIN

```

```

-----
00295      TO PREPARE-TO-RETURN
00296 C     .
00297 C     .   COMPUTE MMORE.
00298 C     .   PROBLEM WILL BE SOLVED ON XBEGIN TO XQUIT WITH MUDED EQUALLY
00299 C     .   SPACED POINTS. USER SHOULD COMPLETE THE INTEGRATION
00300 C     .   FROM XQUIT TO XEND USING AT LEAST MMORE POINTS.
00301 C     .
00302      .   MMORE = 0
00303      .   IF(MUSED.NE.M)
00304      .       .   MMORE = ONE + ABS(XEND-XQUIT)*TEN*ANRMAX
00305      .       .   MMORE = MAXO(7,MMORE)

```



```

00306 . ...FIN
00307 C .
00308 C . RETURN GLOBAL ERROR ELM2 AND R, THE MESH REFINEMENT FACTOR.
00309 C . THE ERROR IS THAT OF THE SOLUTION ON XBEGIN TO XQUIT. TO
00310 C . REDUCE THE ERROR, THE PROBLEM SHOULD BE SOLVED AGAIN ON
00311 C . THIS INTERVAL WITH THE NUMBER OF POINTS BEING MUSED
00312 C . MULTIPLIED BY ONE FACTOR OF R FOR EACH FACTOR OF TEN
00313 C . REDUCTION DESIRED IN THE ERROR.
00314 C .
00315 . IF(MUSED.NE.1)
00316 . . ERROR = ELM2
00317 . . WHEN(ORDER.EQ.0) R=TEN
00318 . . ELSE R=TEN**((ONE/FLOAT(ORDER)))
00319 . ...FIN
00320 ...FIN

```

```

00321 TO GENERATE-A-SOLUTION
00322 C .
00323 C . COMPUTE THE NUMBER OF FORMULAS IN EACH BLOCK.
00324 C . COMPUTE THE END OF THE INTERVAL FOR THE SWEEP.
00325 C .
00326 . NFORMS = MAX0(1,2*(L-1))
00327 . NEND = MUSED - NFORMS
00328 . NTERMS = NFORMS + 1
00329 C .
00330 C . SWEEP THROUGH THE INTERVAL. COMPUTE THE SOLUTION
00331 C . EXCEPT FOR LAST BLOCK.
00332 C .
00333 . DO(I=1,NEND,NFORMS)
00334 . . COMPUTE-A-BLOCK
00335 . ...FIN
00336 C .
00337 C . COMPUTE SOLUTION FOR THE LAST BLOCK.
00338 C .
00339 . I = NEND
00340 . COMPUTE-A-BLOCK
00341 ...FIN

```

```

00342 TO COMPUTE-A-BLOCK
00343 C .
00344 C . COMPUTE IN Y2(*,*) NEW SOLUTION VALUES AT MESH POINTS
00345 C . I+1,...,I+NFORMS.
00346 C . ITERATE FOUR TIMES FOR EACH BLOCK FORMULA STARTING WITH
00347 C . INITIAL GUESSES STORED IN Y2(*,*) FROM PREVIOUS SWEEP.
00348 C .
00349 . DO(KK=1,4)
00350 . . COMPUTE-DERIVATIVES
00351 C .
00352 C . . ITERATE ONCE.
00353 C .
00354 . . DO(J=1,NFORMS)
00355 . . . CALL SCOPY(N,YP(1,1),1,TEMP,1)
00356 . . . CALL SSCAL(N,COEFF(1,J,L),TEMP,1)
00357 . . . DO(IK=2,NTERMS)
00358 . . . . CALL SAXPY(N,COEFF(IK,J,L),YP(1,IK),1,TEMP,1)
00359 . . . ...FIN
00360 . . . FACT = H*DIVSOR(J,L)
00361 . . . CALL SSCAL(N,FACT,TEMP,1)
00362 . . . CALL SAXPY(N,ONE,Y2(1,1),1,TEMP,1)
00363 . . . CALL SCOPY(N,TEMP,1,Y2(1,I+J),1)

```

```

00364      . . . . .FIN
00365      . . . . .FIN
00366      . . . . .FIN

```

```

-----
00367      TO COMPUTE-DERIVATIVES
00368 C      .
00369 C      . EVALUATE DERIVATIVES YP THROUGHOUT THE BLOCK.
00370 C      .
00371 C      . DO(K=1, NTERMS)
00372 C      .
00373 C      . . FORM THE PRODUCT A*Y.
00374 C      .
00375 C      . . DO(J=1, N)
00376 C      . . . YP(J, K) = SDOT(N, A(J, 1, I+K-1), N, Y2(1, I+K-1), 1)
00377 C      . . . . .FIN
00378 C      .
00379 C      . . ADD IN G VECTOR.
00380 C      .
00381 C      . . CALL SAXPY(N, ONE, G(1, I+K-1), 1, YP(1, K), 1)
00382 C      . . . . .FIN
00383 C      . . . . .FIN

```

```

-----
00384      TO COMPUTE-GLOBAL-ERROR
00385 C      .
00386 C      . THE ERROR IN SOLUTION Y1(*, *) IS ESTIMATED BY COMPARING
00387 C      . IT TO THE MORE ACCURATE SOLUTION Y2(*, *).
00388 C      . THE ERROR MEASURED IS ABSOLUTE IF THE COMPONENT IS LESS
00389 C      . THAN ONE IN MAGNITUDE, OTHERWISE RELATIVE.
00390 C      . THE GLOBAL ERROR ELM1 IS THE LARGEST ERROR SEEN IN ANY
00391 C      . COMPONENT IN THE WHOLE INTEGRATION.
00392 C      .
00393 C      . ELM1 = ZERO
00394 C      . DO(K=2, M)
00395 C      . . DO(I=1, N)
00396 C      . . . EMAX = AMAX1(ONE, ABS(Y2(I, K)))
00397 C      . . . EMAX = ABS(Y2(I, K) - Y1(I, K)) / EMAX
00398 C      . . . ELM1 = AMAX1(ELM1, EMAX)
00399 C      . . . . .FIN
00400 C      . . . . .FIN
00401 C      . . . . .FIN
00402      END

```

Distribution:

C. W. Gear
Dept. of Computer Science
University of Illinois
Urbana, Illinois 61801

T. E. Hull
Dept. of Computer Science
University of Toronto
Toronto, Canada M5S 1A7

J. V. Wait
Dept. of Electrical Engineering
University of Arizona
Tucson, Arizona 85721

S. Cohen (5)
Speakeasy Center
Argonne National Laboratory
Argonne, IL 60439

G. D. Byrne
Mathematics Department
University of Pittsburgh
Pittsburgh, PA 15260

J. C. T. Pool
Engineering, Mathematical and
Geo-Sciences
Division of Basic Energy Sciences
U.S. Energy Research and Development
Administration
Washington, D.C. 20545

5000 A. Narath
5100 J. K. Galt
5110 F. L. Vook
5120 G. J. Simmons
5121 R. J. Thompson
5122 L. F. Shampine (10)
5130 G. A. Samara
5150 J. E. Schirber
5160 W. Herrmann
5200 E. H. Beckner
5400 A. W. Snyder
5700 J. H. Scott
5800 R. S. Claassen
8266 E. A. Aas (2)

3141 C. A. Pepmüller (5)
3151 W. L. Garner (3)
For ERDA/TIC (Unlim. Release)
3171-1 R. Campbell (25)
For ERDA/TIC