27
3-28-80
SAND80-0429
Unlimited Release
UC-32

24 DUTIS

Linear Equations in General Purpose Codes for Stiff ODEs

MASTER

Lawrence F. Shampine

Prepared by Sandia Laboratories, Albuquerque, New Mexico 87185
and Livermore, California 94550 for the United States Department
of Energy under Contract DE-AC04-76DP00789

Printed February 1980

Sandia Laboratories

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
Department of Energy 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 Department of Energy, 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.

SF 1004-DF(11-77)

Printed in the United States of America

Available from

National Technical Information Service

U. S. Department of Commerce

5285 Port Royal Road

Springfield, VA 22161

Price: Printed Copy \$4.00; Microfiche \$3.00

SAND80-0429
Unlimited Release
Printed February 1980

UC-32

Linear Equations in General Purpose
Codes for Stiff ODEs

Lawrence F. Shampine
Numerical Mathematics Division 5642
Sandia Laboratories
Albuquerque, New Mexico 87185

Introduction

F. T. Krogh once remarked to the author that the more one knew about an ordinary differential equation (ODE) and its solution, the better he could solve it numerically. This is certainly true, but it carries the proviso that a special treatment be worth the trouble. The general purpose codes for nonstiff problems are so effective that it is unusual for a special code to be justified. The situation is somewhat different for stiff problems. For them a user is asked to recognize and present his problem in such a way that several subclasses are given special treatment. He must, in the first place, indicate whether the problem is stiff - a difficult decision. He then must decide whether to supply a subroutine for evaluating the Jacobian of the differential equation or to ask the code to form the Jacobian by numerical differentiation. He must also describe the structure of the Jacobian - dense, banded, or sparse - and he might be put to considerable trouble to store the Jacobian appropriately.

When considering whether to single out a class of problems for special treatment, there are a number of factors to be considered: Is the class of problems important? How hard is it for the user to recognize that his problem is in the class? How much better can the code do if it knows about the special nature of the problem? How much work is required of the user to solve such problems?

Linear problems have many special properties which have long been studied. In some contexts linear problems are quite important in practice, and there are even special codes for them. Some years ago the author [1] noted a software engineering matter which made general purpose codes considerably more efficient when applied to (nonstiff) linear problems. The author's colleagues M. R. Scott and H. A. Watts have been developing software for the solution of linear boundary value problems by shooting methods. H. A. Watts [2] found that the author's idea, and one of his own also making use of linearity, significantly increased the efficiency of solution of the boundary value problems by general purpose codes for the initial value problem.

Thus linear differential equations are important, extremely easy to recognize, and it has been shown that advantage can be taken of linearity in general purpose codes for nonstiff problems. When the author and C. W. Gear were writing the survey [3] on the solution of stiff problems, they talked about how effective general purpose codes are when confronted with a

linear problem. The conclusion that they do well was interpreted to mean that nonlinearity is handled rather well. Recently the author had occasion to ask a slightly different question. Is it possible to improve significantly the handling of linear problems in a general purpose code with very little trouble to the user or change to the code? In this report we answer the question in the affirmative. Our proposals are applicable to a great variety of codes, but when we wish to be specific, we shall refer to a typical implementation of the backward differentiation formulas (BDF) as in Gear's famous code DIFSUB [4] or one of its many reincarnations and relatives.

The Proposals

The first, and perhaps most important, proposal is a software engineering issue rather than an algorithmic issue. The general form of the system of differential equations is

$$(1) \quad y' = f(x, y) \quad , \quad y(a) \text{ given} \quad .$$

Practically all codes intended for stiff problems ask the user whether he is providing a subroutine to evaluate the Jacobian of f analytically. If he is not, the code approximates the Jacobian by numerical differencing. Differencing is so convenient for the user that it is exceedingly popular. It may or may not be cheaper than analytical evaluation of the partial derivatives of f . Because of scaling difficulties it may result in a poor approximation to the Jacobian with attendant difficulties for the code.

When the equations (1) are linear,

$$(2) \quad y' = g(x)y + g(x) \quad ,$$

the situation is special in several important ways. The Jacobian matrix is $g(x)$. The codes presume that evaluation of the Jacobian is quite expensive compared to an evaluation of the function f . For a linear problem it is actually cheaper. If one were to write a special code, he could even take advantage of the fact that $g(x)$ is evaluated every time f is, but we are concerned only with general purpose codes and do not propose this.

The typical differencing algorithm forms the Jacobian a column at a time by

DISCLAIMER

This book 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.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

$$\frac{\partial f}{\partial y_j} = \frac{f(x, y + \delta e_j) - f(x, y)}{\delta}$$

where e_j is the j^{th} column of the identity matrix and δ is a suitable scalar. Using some evaluation of the vector f at (x, y) , the whole Jacobian can be approximated with n extra function evaluations for a system of n equations. This is the situation for a dense Jacobian. The ingenious idea of [5] reduces the number of extra evaluations when the Jacobian has a known sparsity structure. For example, if the Jacobian is known to be banded and of width $2m+1$, it can be formed in only $2m+1$ extra evaluations of f .

It is clear that if the problem is linear, numerical differencing is a very expensive way to generate the Jacobian. It is, e.g., more than n times the cost of an analytical evaluation in the dense case. In view of the fact that the user must program the evaluation of $g(x)$ in order to do the evaluation of f which is required of him, it cannot ordinarily be much trouble for him to supply a routine for evaluating the Jacobian. He might even want to program the subroutine for f so that it first calls the Jacobian subroutine and then completes the evaluation of f .

The documentation of the code should make it clear that if the system of equations is linear, it is little trouble to supply a routine for the Jacobian, it is more reliable, and it is a lot cheaper. This way of urging the user to handle linear problems properly was adopted in the user interface of Shampine and Watts for DEPACK. Of course it is an easy thing to do in any code for stiff problems. In the author's experience users must be urged strongly and some reason given for them supplying an analytical Jacobian, or they will not even consider the matter.

The main thing we ask of the user is whether his problem is linear. It is very little extra trouble for him to say if the Jacobian is constant as well. This is quite a special case, but not uncommon in some contexts. As we shall see, one can greatly improve the solution of such problems. We are aware of one code, COLODE [6], which does ask if the Jacobian is constant. Curiously it does not take advantage of linearity, although it could by the means we describe below. What we have in mind requires an exact Jacobian, so we propose using an indication of linearity only when the user also accepts our advice to supply an analytical Jacobian.

Methods for solving stiff ODE problems are implicit. A form typical of of formulas such as the BDF is

$$(3) \quad y_{n+1} = h\gamma f(x_{n+1}, y_{n+1}) + \psi_n$$

where y_{n+1} is to be the approximation of $y(x_{n+1})$, h is the step size, γ is a constant characteristic of the formula, and ψ_n lumps together previously computed quantities. Such a system of nonlinear algebraic equations is ordinarily solved by the iteration (simplified Newton method)

$$(4) \quad (I - h\gamma J) \left(y_{n+1}^{(m+1)} - y_{n+1}^{(m)} \right) = - \left(y_{n+1}^{(m)} - h\gamma f(x_{n+1}, y_{n+1}^{(m)}) - \psi_n \right)$$

Here J is an approximation to the Jacobian evaluated at (x_{n+1}, y_{n+1}) . The codes expect formation of J to be expensive, so they try to do this no more often than necessary.

If the ODE is linear, so is the algebraic problem (3) and the linearization of (4) is apparently pointless. The factorization of the iteration matrix $I - h\gamma J$ in (4) is relatively expensive. The codes avoid changing the step size h and the method (here showing up in γ) unless it is necessary or a significant increase in h is possible. In this way the same factored iteration matrix can be used on succeeding steps for as long as convergence of the iteration (4) is rapid. This procedure is more efficient than solving the linear algebraic equations (3) directly when the ODE is linear. Still, from time to time a new Jacobian and iteration matrix must be formed and the iteration matrix factored. If the problem is linear, the problem (3) is solved exactly in just one iteration for this step. Of course this is true at every step when the Jacobian is constant.

Solution of (3) proceeds as follows: From previously obtained results, a prediction $y^{(0)}$ is made. Then the right-hand side of (4) is evaluated. Next the linear system (4) is solved for $y^{(1)} - y^{(0)}$, the correction to the predicted value. If the iteration matrix is formed from the correct Jacobian, the iteration could be terminated at this point. This is what we propose to do. All the code needs to be told is that the problem is linear and that a subroutine for the Jacobian has been provided. The code just keeps track whether $J = J(x_{n+1})$. Without this change the code would form $y^{(1)}$, and again evaluate the right-hand side of (4). If a residual convergence test is made, as recommended by Shampine [7] and Williams [8], the iteration will terminate then with a zero residual. If a typical convergence test is made, (4) must again be solved to compute $y^{(2)} - y^{(1)}$. The

iteration will terminate then with zero correction.

When the special situation described happens, the number of function evaluations is halved. If a usual convergence test is made, the number of solutions of linear systems is also halved. The situation occurs at every step if the Jacobian is constant. We have to look carefully at what the codes do when the Jacobian is not constant.

One might think that every time the Jacobian is evaluated and the iteration matrix $I - h y J$ formed, the formula (3) is solved exactly in one iteration of (4). Unfortunately this is not so. What happens precisely depends on the code. In DIFSUB and its relatives, let us suppose the solution has reached (x_n, y_n) and that the algorithm chooses to increase the step size and possibly change the formula. The call to the Jacobian subroutine is made with the argument (x_n, y_n) . The argument one really wants is (x_{n+1}, y_{n+1}) but, of course, y_{n+1} is not known. When solving stiff problems one hypothesizes that the Jacobian is slowly varying on and near the solution $y(x)$ so that the argument used is not unreasonable. We do not want to discuss here what might or should be done in general. In the case of linear problems the matter is clearer. The correct value of x is x_{n+1} and y is only a dummy vector. We propose that the correct argument x_{n+1} be used for linear problems. This is a minor change to the code but results in the correct Jacobian whenever it is evaluated. Furthermore it is a slightly more realistic approximation to the Jacobian for subsequent steps. If the step to $x_n + h$ fails, the line of codes being discussed will reevaluate the Jacobian at (x_n, y_n) . This is the same evaluation done earlier, but it is repeated so as to avoid storing a Jacobian. Naturally we propose instead to evaluate at $x_n + h'$ where h' is the new step size to be attempted. Thus in no case does our proposal result in extra work and it provides a more accurate Jacobian for linear problems. With this change, the observation about reducing the number of function evaluations and solution of linear systems is applicable to this line of codes whenever a new iteration matrix is formed.

Some authors [9,10,11] recommend retaining a copy of the approximate Jacobian and judiciously reusing it when forming a new iteration matrix. For example, we have noted that in the line of codes starting from DIFSUB, the same Jacobian may be evaluated repeatedly when a step from x_n fails repeatedly. According to this idea, the stored Jacobian would be used instead. Justification of the idea fails when the Jacobian is extremely cheap. Thus we propose

a different tactic for linear problems. For them one should probably evaluate the Jacobian any time a new iteration matrix is formed and factored. This provides a better quality approximation, which it is hoped will prove more efficient, and evaluates the Jacobian at the right place for at least one step. It should be easy to alter the tactics for this special case in most codes.

Comments on Testing

One objective of the influential set of stiff test problems of Enright et al. [12] is to study the effect of nonlinearity. For this reason the set of 25 problems includes 9 linear problems. All the linear problems have constant Jacobians. One, A2, is a "real" problem due to Klopfenstein and Davis. The rest were constructed to have certain spectral properties. So many problems with constant Jacobian is obviously a technical convenience and not a reflection of their frequency of occurrence.

There is an unfortunate consequence of all the linear problems having a constant Jacobian. We have pointed out that for a general linear problem, the typical code does not evaluate the Jacobian at the right argument, so that solution of (3) by (4) is not exact in one iteration. However, the solution is exact when the Jacobian is constant. For this reason the performance of the codes tested cannot be regarded as truly representative of their behavior for a general linear problem. Undoubtedly this was not the intent of the authors of [12]. The behavior for general linear problems must be rather different. More iterations are surely necessary and convergence is not recognized so easily. Indeed, problems of this kind must present the convergence test with the easiest decision it will ever have to make.

The tests of Enright et al. were not intended to mirror computing practice. For example, the codes were altered or written to invert iteration matrices rather than factor them. Another example is that analytical Jacobians were supplied. This is useful for suppressing the effects of the various differencing algorithms in order to appraise other algorithms in the codes, but we do not think it reflects typical use of the codes.

The "real" linear problem A2 has 9 equations. With solved with a version of DIFSUB at a pure absolute error tolerance of 10^{-4} , it is reported in [12] that 306 function evaluations, 19 evaluations of the Jacobian, 19 inversions of the iteration matrix, and 121 steps were made. Here a Jacobian

evaluation is essentially the same as a function evaluation because the equations are homogeneous. Enright et al. solved the problem with an analytical Jacobian as we recommend. Had they used differences and treated the Jacobian as dense, $9 \times 19 = 171$ additional function evaluations would have been made. We think this is a common way to proceed and it is obviously very expensive. The Jacobian happens to be tridiagonal. Even if the code had provided for an appropriate differencing scheme and if the user had told the code of the structure, the cost would have been $3 \times 19 = 57$ extra function evaluations. If one had altered the code to recognize linearity, a function evaluation and the solution of a linear system would be saved every time an iteration matrix was formed - here whenever a Jacobian was formed. This would have been a significant savings. The problem has a constant Jacobian. If one had altered the code to recognize this very special case, the number of function evaluations would have been halved, as would the number of times a linear system was solved.

It is always true that numerical tests must be interpreted with the greatest caution so as to understand as fully as possible the relation of the codes to the tests and the criteria for evaluation. For example, it is of great practical importance whether a code inverts iteration matrices or factors them and whether it provides for Jacobians with structure other than dense. The figures we have given make it clear that application of codes altered along the lines we suggest to the test set [12] must be interpreted very carefully. Naturally one wants to take advantage of linearity and of constant Jacobians, but he must not think that this set was constructed to represent fairly the frequency of occurrence of such problems.

References

1. L. F. Shampine, Cheaper integration of linear systems, *Simulation*, 20 (1973) 17.
2. H. A. Watts, Efficient integration processes for solving boundary value problems, Rept. SAND78-1244, Sandia Laboratories, Albuquerque, NM, 1978.
3. L. F. Shampine and C. W. Gear, A user's view of solving stiff ordinary differential equations, *SIAM Review*, 21 (1979) 1-17.

4. C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, NJ, 1969.
5. A. R. Curtis, M. J. D. Powell, and J. K. Reid, On the estimation of sparse Jacobian matrices, J. Inst. Math. Appl., 13 (1974) 117-119.
6. B. L. Hulme and S. L. Daniel, COLODE: a collocation subroutine for ordinary differential equations, Rept. SAND74-0380, Sandia Laboratories, Albuquerque, NM, 1974.
7. L. F. Shampine, Evaluation of implicit formulas for the solution of ODEs, BIT, to appear.
8. J. Williams, The problem of implicit formulas in numerical methods for stiff differential equations, Rept. 40, Dept. of Math., University of Manchester, Manchester, England, 1979.
9. L. F. Shampine, Implementation of implicit formulas for the solution of ODEs, SIAM J. Scientific, Stat. Comp., to appear.
10. A. R. Curtis, Solution of large, stiff initial value problems - the state of the art, pp. 257-278 in D. Jacobs, ed., Numerical Software - Needs and Availability, Academic Press, New York, 1978.
11. T. Chambers, The use of numerical software in the digital simulation language FMSP, pp. 237-253 in D. Jacobs, ed., Numerical Software - Needs and Availability, Academic Press, New York, 1978.
12. W. H. Enright, T. E. Hull, and B. Lindberg, Comparing numerical methods for stiff systems of ODEs, BIT, 15 (1975) 10-48.

Distribution:

Dr. F. T. Krogh
125/128
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91103

Dr. R. W. Klopfenstein
RCA Laboratories
Princeton, NJ 08540

Prof. D. G. Bettis
TICOM
University of Texas at Austin
Austin, TX 78712

Prof. C. W. Gear
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Prof. R. D. Skeel
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Dr. A. C. Hindmarsh
L-310
Lawrence Livermore Laboratory
Livermore, CA 94550

Dr. Linda Petzold
Division 8332
Sandia Laboratories
Livermore, CA 94550

Prof. T. A. Bickart
Elec. and Comp. Engr. Dept.
Link Hall
Syracuse University
Syracuse, NY 13210

Prof. G. D. Byrne
Dept. of Mathematics and Statistics
University of Pittsburgh
Pittsburgh, PA 15260

Prof. T. E. Hull
Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 1A7

Dr. W. H. Enright
Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 1A7

Prof. F. Chipman
Dept. of Mathematics
Arcadia University
Wolfville, Nova Scotia
CANADA BOP 1X0

Dr. J. P. Hennart
I.I.M.A.S.
Universidad Nacional Autónoma
de México
Apartado Postal 20-726
México 20, D.F.
MEXICO

Dr. R. England
I.I.M.A.S.
Universidad Nacional Autónoma
de México
Apartado Postal 20-726
México 20, D.F.
MEXICO

Prof. G. Dahlquist
Dept. of Numerical Analysis
Royal Institute of Technology
S-10044 Stockholm
SWEDEN

Dr. S. P. Nørsett
Universitet i Trondheim
Norges Tekniske Høgskole
Institutt for Numerisk Matematikk
N 7034 Trondheim
NORWAY

Prof. J. D. Lambert
Department of Mathematics
University of Dundee
Dundee DD1 4HN
SCOTLAND

Dr. George Hall
Department of Mathematics
University of Manchester
Manchester M13 9PL
ENGLAND

Dr. I. Gladwell
Department of Mathematics
University of Manchester
Manchester M13 9PL
ENGLAND

Dr. A. Prothero
Thornton Research Centre
P. O. Box No. 1
Chester CHI 3SH
ENGLAND

Prof. Dr. H. J. Stetter
Institut für Numerische Mathematik
Technische Hochschule Wien
A-1040 Wien, Gusshausstr. 27-29
AUSTRIA

Dr. R. Sacks-Davis
16 Cantala Ave.
Caulfield
Victoria 3161
AUSTRALIA

Prof. Dr. P. Deuflhard
Universität Heidelberg
Institut für Angewandte Mathematik
Im Neuenheimer Feld 294
6900 Heidelberg 1
WEST GERMANY

Prof. J. C. Butcher
Department of Mathematics
University of Auckland
Auckland
NEW ZEALAND

Prof. G. K. Gupta
Division of Computer Applications
Asian Institute of Technology
P. O. Box 2754
BANGKOK, THAILAND

2623 M. R. Scott
2623 H. A. Watts
5600 D. B. Shuster
Attn: 5610 A. A. Lieber
5620 M. M. Newsom
5630 R. C. Maydew
5640 G. J. Simmons
5641 R. J. Thompson
5642 L. F. Shampine (5)
8266 E. A. Aas
8332 R. E. Huddleston
8332 L. Petzold
3141 T. L. Werner (5)
3151 W. L. Garner (3)
For DOE/TIC (Unlim. Release)
3154-3 R. P. Campbell (25)
For DOE/TIC)