Vita

please look at last page

J.T. Oden, Ed. COMPUTATIONAL METHODS IN NONLINEAR MECHANICS

CHAPTER 19

LIPSCHITZ CONSTANTS AND ROBUST ODE CODES*

L.F. SHAMPINE

Applied Mathematics Research Department, Sandia Laboratories, Albuquerque, NM 87185, U.S.A.

Estimates of local Lipschitz constants can be computed cheaply with Adams and Runge–Kutta methods. ODE codes can be made more robust with this information by helping the user know if his problem is correctly formulated, if he is using the right kind of method, and if the method is being applied in a reliable way.

19.1. Introduction

In this paper we show how to estimate cheaply a local Lipschitz constant when solving an initial value problem for an ordinary differential equation (ODE) with Adams or Runge–Kutta methods. We are most concerned about the occurrence of a 'large' Lipschitz constant. This may mean that the problem is not correctly posed. It certainly means that the solution will be expensive. There is a class of such problems called *stiff* which can be solved efficiently by other methods. We consider how to recognize such problems. The Adams and Runge–Kutta methods are extremely effective for the solution of non-stiff problems. Their reliability depends upon the code having some idea as to the scale of the problem. We consider how to achieve this using an estimate of the Lipschitz constant. Thus we propose to obtain information which will help the user know that his problem is correctly formulated, that he is using the right kind of method, and that the method is being applied in a reliable way. Capabilities of this kind are part of what is called the robustness of a code.

An example from the literature illustrates the practical value of the

* This article sponsored by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

capabilities proposed. Scott and Watts [1976] report their solution of a boundary value problem arising in the modelling of a kidney. There are five equations for which four initial values are given at one point. A shooting method was used whereby the fifth initial value at this point was guessed, the resulting initial value problem integrated, and the guessed initial value adjusted depending on how well the known value at the far end of the interval was approximated. The very efficient variable order Adams code ODE of Shampine and Gordon [1975] was used. Starting with the guess $y_5(0) = 0.99026$, the integration to 1 cost about 0.88 seconds of computing time on a CDC 6600. With $y_5(0) = 0.99000$ the cost soared to 101.5 seconds! The second integration was accomplished in 1.08 seconds by a code aimed specifically at stiff problems. The authors explored values of $y_5(0)$ ranging from 0 to 100 and only in some ranges is the problem stiff. The extreme stiffness is not apparent even to an expert and anyway, the integrations take place in a module out of sight. Actually ODE has a test for stiffness; Scott and Watts report that when they looked into the matter, they found the code had indicated 81 times in the expensive run that the problem was stiff. The ideas developed in this paper provide a much more effective test for ODE.

19.2. Mathematical preliminaries

We are interested in the numerical solution of the initial value problem

$$y' = f(x, y), \quad y(a) \text{ given}, \quad a \le x \le b, \tag{19.1}$$

where y and f are n-vectors. We say f satisfies a Lipschitz condition with constant L on a region R if for a specific norm

$$\|f(x, u) - f(x, v)\| \le L \|u - v\| \tag{19.2}$$

for all (x, u) and (x, v) in R. In future we shall refer to 'the' Lipschitz constant and mean the smallest constant L which satisfies (19.2). Supposing that (19.1) has a unique solution $y(x)$, we need only concern ourselves with the behavior of f near $(x, y(x))$. Presuming that f has continuous first partial derivatives on such a neighborhood, the mean value theorem states that

$$f(x, u) - f(x, v) = J(u - v).$$

Here J is the Jacobian matrix

$$J = \left(\frac{\partial f_i}{\partial y_j}\right)$$

with entries evaluated at (in general) different points along the line between u and v. Thus 'the' Lipschitz constant is the maximum of $\|J\|$ on the neighborhood and for a small neighborhood

$$\|J(x, y(x))\| \doteq L. \tag{19.3}$$

In the rest of this paper, when we refer to a Lipschitz constant, we have in mind that (19.2) holds in a small neighborhood of the curve $(x, y(x))$ and that (19.3) is a good approximation there.

If $u(x)$ and $v(x)$ are two solutions of the differential equation, then

$$\|u(x + h) - v(x + h)\| \le \|u(x) - v(x)\| \exp(hL). \tag{19.4}$$

This tells us how fast solution curves can spread apart, but it does not give us a very good qualitative picture because it does not distinguish between problems for which the curves converge and those for which they diverge. The eigenvalues of the Jacobian are more informative. If an eigenvalue has positive (negative) real part, some solution curves can diverge (converge) and the larger the real part, the faster this happens. The spectral radius, $\rho(J)$, is the maximum magnitude of the eigenvalues of the Jacobian. The Lipschitz constant and the inequality (19.4) depend on the norm, and the eigenvalues do not. However, in any norm

$$\rho(J) \le \|J\|,$$

which helps show the connection between the two points of view.

19.3. Why estimate the Lipschitz constant?

We are interested in the numerical solution of ordinary differential equations by classical methods such as Adams and Runge–Kutta. The inequality (19.4) describes a fundamental limitation of such step-by-step methods. An error at any point x can grow by a factor of $\exp(hL)$ in a step of length h even if the numerical method introduces no other error. If this inequality is realistic and L is 'large' compared to the interval of integration $[a, b]$, only a limited accuracy is possible. Such problems are called ill-posed or unstable. The practical limitation is vague because it depends on the precision available for the computation and the number of steps (the work) one is willing to expend. The bound presumes solution

curves diverge as rapidly as possible. Most physical problems are well-posed so that regions where curves diverge slowly are balanced by regions where they converge.

If a 'large' Lipschitz constant is encountered, it should worry the poser of the problem. One possibility is that the problem is genuinely unstable. This is almost surely the result of a mistake in the coding of the differential equation or in the formulation of the model leading to the equation. Another possibility is that the problem is very stable. In this case the solution is going to be expensive. Fortunately a great many such problems arising from physical situations fall into a special class called *stiff*. Special methods are quite efficient for these problems. Thus, recognizing a 'large' Lipschitz constant provides important information about the problem. Being able to recognize such a problem as stiff provides important information about the appropriateness of classical numerical methods.

The fundamental possibility of error growth in one step as fast as $\exp(hL)$ implies that the quantity hL must be controlled in size. In fact, all the basic algorithms of the numerical methods require that h be 'sufficiently' small in order that asymptotic arguments be valid. Unfortunately we do not know how to quantify this realistically. The numerical method will evaluate the equation only at a discrete set of points. If a considerable change in the solution is possible between these points, we cannot hope to approximate it reliably. The Lipschitz constant tells us how fast the solution can change, so hL being 'small' expresses our intuitive feeling about the situation. The algorithms are designed to cope with slow changes as the integration proceeds. Usually they will do well if the first step is on scale. At crude accuracies the accuracy requirement may not restrict the step size sufficiently and the code may begin to skip over important changes in the solution. These remarks about the necessity of controlling hL suggest that an estimate of L will enhance the reliability of the computation, especially when little accuracy is needed.

Having indicated in general terms the value of estimating the Lipschitz constant, we shall spend the rest of this section being more quantitative about the applications which we propose.

It is impossible to be precise about what constitutes a 'large' Lipschitz constant. We have already noted the role of the interval in the possible divergence of solution curves. Another way to see the effect of the interval $[a,b]$ is to scale the independent variable so as to solve the problem on a standard interval $[0,1]$. The effect of scaling is to multiply the Lipschitz constant and the eigenvalues by $(b-a)$. Clearly then, 'large' must mean

that $(b-a)L$ is 'large'. People do solve moderately unstable problems. The method of shooting solves boundary value problems via a set of initial value problems which are often unstable. They are solved by overpowering the difficulty with a lot of precision and work and by reducing the interval of integration through the idea of multiple shooting. People also solve a great many stiff problems, which have 'large' Lipschitz constants but which are stable. We shall discuss stiff problems in a moment. Right now we note that classical methods should be replaced by alternatives in this case. There is a considerable amount of practical experience as to when the alternatives are more effective. A reasonable choice for a definition of large Lipschitz constant L is one for which $|b-a|L \geq 500$. If the problem is unstable, this certainly represents very serious sensitivity. If the problem is stable but not stiff, restrictions on hL will mean it is expensive to solve. If the problem is stable and stiff, one should turn to other methods. Thus if we find such an L, the user should examine his equation and model. We shall try to discover automatically if the problem is stiff and report it separately.

A general discussion of stiff problems can be found in Shampine and Gear [1979]. Such problems are stable and have slowly varying, smooth solutions. Classical numerical methods have stability regions R such that $h\lambda_i \in R$, for the step size h and all eigenvalues λ_i of the Jacobian J which have $\mathrm{Re}(\lambda_i) \leq 0$, is necessary for the stability of the numerical method. Stiff problems are those stable problems which have at least one eigenvalue with $\mathrm{Re}(\lambda_i) \ll -1$ and also have a solution which changes slowly over long intervals. On an interval of slow variation, the solution could be approximated accurately with a large step size, but the stability requirement forces a small step size to be used. There exist methods with infinite stability regions. They are so costly per step that they are advantageous only when the stability restriction is severe. The typical stiff problem does have intervals in which the solution changes rapidly and the requirement of accuracy demands a small step size, but most of the integration can be accomplished much more efficiently with a special method without stability limitations.

Error might grow as fast as $\exp(hL)$ in the course of a step. Without the eigenvalues of the Jacobian (ordinarily it is far too expensive to compute them), we are not sure how realistic this is. Experience says that the typical non-stiff problem is moderately stable with intervals of both mild divergence and convergence. For this reason we should not be too pessimistic, but it would be rash to let hL get very big. This lack of information is typical as regards the control of hL. It is clear we dare not let it get big, but

it is hard to say just how big is permissible. We shall take up a few situations where we can be more concrete.

The numerical experiments we report in Section 19.6 were made with the Adams code ODE. Because of this and so as to be clearer, we shall provide details of the method of order 1 in this code. This method is always used for the first step so limitations on its step size which help assure one is on scale are quite pertinent to the performance of the code. The method uses an approximate solution value y_n at x_n and the slope $f_n = f(x_n, y_n)$ to predict a solution at x_{n+1} by

$$p_{n+1} = y_n + hf_n, \qquad f^p_{n+1} = f(x_{n+1}, p_{n+1}).$$

It then forms two solution values of different orders of accuracy:

order 1: $\bar{y}_{n+1} = y_n + hf^p_{n+1}$,

order 2: $y_{n+1} = y_n + \frac{1}{2}h[f_n + f^p_{n+1}]$.

The local error of the result of order 1,

$$\text{local error} = y(x_{n+1}) - \bar{y}_{n+1},$$

is estimated by comparison to the result of order 2,

$$\text{est} = y_{n+1} - \bar{y}_{n+1}.$$

The code actually uses the more accurate result y_{n+1} to continue the computation (local extrapolation).

A group of workers at the University of Toronto have been studying the application of Runge–Kutta methods to the model problems

$$y' = Jy, \quad y(a) \text{ given}, \quad J \text{ a matrix.} \tag{19.5}$$

Problems in this class are so simple that one can compare the true local errors and the estimated local errors in considerable detail. The paper of Jackson et al. [1978] emphasizes conclusions about efficiency. However, one of the things they do is to ask what restriction on L is necessary to be sure that $\|\text{local error}\| \le \|\text{est}\|$. This is a reliability question because if the inequality does not hold, the code may make a mistake about whether to accept the step. We prefer to think of such results as necessary conditions for reliability. The Toronto group also uses them for this purpose. In several codes, e.g., Hull et al. [1976], they ask the user to supply a guess for L in order to restrict h properly. We shall estimate L so our investigation is in part making convenient and automatic this approach to reliability. The kind of restriction arising here will be illustrated by a simplified analysis we developed for the method in ODE.

When the method of order 1 in ODE is applied to a problem in the class of (19.5) it is easily found that

$$\text{local error} = [\exp(hJ) - (I + hJ + \tfrac{1}{2}(hJ)^2)]y(a),$$

$$\text{est} = \tfrac{1}{2}(hJ)^2 y(a).$$

We can rewrite the local error expression as

$$\text{local error} = 2[(hJ)/3! + (hJ)^2/4! + \cdots]\tfrac{1}{2}(hJ)^2 y(a).$$

The series

$$S(\|hJ\|) = 2[\|hJ\|/3! + \|hJ\|^2/4! + \cdots]$$

converges uniformly for all $\|hJ\|$ which implies that

$$\|\text{local error}\| \le S(\|hJ\|)\|\text{est}\|.$$

From the expression

$$S(x) = \frac{2[\exp(x) - (1 + x + \frac{1}{2}x^2)]}{x^2}$$

we find easily that $S(\|hJ\|) \le 1$ if $\|hJ\| \le c$ for a constant c about 1.79. The bounds here are sharp, for if $\|hJ\| > c$, there is a problem in the class for which $\|\text{local error}\| > \|\text{est}\|$. For this class $\|J\| = L$ so the condition for reliability is $hL \le c$. This is the kind of result that can be established for other one-step methods with the constant c being determined by the method.

Fundamental limits on the accuracy possible in a given precision are discussed in Shampine [1974]. Let us develop another such limit involving the Lipschitz constant. Our computer cannot form p_{n+1} exactly. We get instead a rounded value $p^*_{n+1} = p_{n+1} + \zeta$, where $\|\zeta\| \le u\|p_{n+1}\|$ and u is the unit of round-off on the machine used. (This is the right kind of bound; the error might be a lot larger.) Even if all other sources of error are ignored, we would compute F^p_{n+1} instead of f^p_{n+1}, where the difference could be as large as

$$\|F^p_{n+1} - f^p_{n+1}\| = L\|p^*_{n+1} - p_{n+1}\| = Lu\|p_{n+1}\|.$$

This means we could make an error in the evaluation of 'est' as large as $\frac{1}{2}hLu\|p_{n+1}\|$. If we are attempting to control the error relative to the solution, this observation says that it is hopeless to try to get accuracies less than $\frac{1}{2}hLu$. In general the Adams methods used in ODE have an error estimator of $h\gamma^*_k\nabla^k f^p_{n+1}$ when working at order k and constant step size h. One then finds that relative errors less than $hLu\gamma^*_k$ cannot be obtained

reliably because we cannot believe the estimate of the error. This is not a stringent limitation, but high accuracy computations are common with a code like ODE and some control of hL is implied.

Another kind of restriction on hL arises from the use of implicit methods. The Adams–Moulton formulas are used in ODE as correctors and the computations are explicit. However, in codes such as Gear's [1971] they are implicit. At each step one must solve an algebraic equation for the next solution value. An example is

$$y_{n+1} = y_n + \tfrac{1}{2}h[f_n + f(x_{n+1}, y_{n+1})].$$

Simple iteration,

$$y_{n+1,m+1} = y_n + \tfrac{1}{2}h[f_n + f(x_{n+1}, y_{n+1,m})],$$

is used for this purpose. It is necessary that $\frac{1}{2}h\rho(J) < 1$ for this process to converge for all starting guesses $y_{n+1,0} = p_{n+1}$. This condition is not realistic in practice because the process can diverge in specific norms for an arbitrary number of iterations and the ultimate convergence can be arbitrarily slow. In practice the convergence must be very fast, and the process must be seen to be converging numerically in the desired norm. Thus in practice one insists that, say, $\frac{1}{2}h\|J\| \leq \frac{1}{3}$ so as to have a contraction by a factor of at least $\frac{1}{3}$ in a neighborhood of y_{n+1}. This means here that $hL \leq \frac{2}{3}$. Similar restrictions have to be imposed for all the implicit methods.

We have exhibited restrictions on hL of a diverse nature that must be imposed in order to solve non-stiff problems reliably. The precise value does not appear to matter too much nor is the restriction at all serious in practice. We defined a large Lipschitz constant as one for which $(b-a)L \geq 500$. One aspect of the choice of 500 was that if we insist that $hL \leq 1$, we can integrate a smooth, slowly varying solution in about $(b-a)L$ steps of length $1/L$. Five hundred steps was found from experience to be a substantial amount of work with ODE. To warn the user that the problem is an expensive one the code already returns with a flag set after taking this many steps. Thus the restriction $hL \leq 1$ is a mild one and is no restriction at all for the typical non-stiff problem at even modest accuracies.

19.4. Assessing the Lipschitz constant

For our purposes we do not need an accurate value for the Lipschitz constant. Just knowing whether or not it is 'large' would be quite useful.

This situation is fortunate because we also do not want to go to very much work to approximate the constant. It is appropriate to consider two circumstances. At the beginning of the computation a little extra work is justified because of the need to select an initial step size which is on scale. Thereafter the algorithms for adjusting the step size will do a creditable job of staying on scale. Furthermore, even a crude estimate of the Lipschitz constant on subsequent steps should detect any sudden, large change which might affect the reliability of the computation. For these reasons only a minimal amount of work will be permitted after the initialization of the code.

19.4.1. *Beginning the integration*

From the definition (19.2), a lower bound for the local Lipschitz constant L can be found by selecting any particular perturbation vector $\delta y \neq 0$ and computing

$$\frac{\|f(x, y+\delta y) - f(x, y)\|}{\|\delta y\|} \leq L. \tag{19.6}$$

The quantity on the left measures the derivative of f in the direction of δy. If we were to explore n directions in this way, we could approximate the Jacobian J and from its norm obtain a suitable value for L. Codes intended for stiff problems do form J and this is one reason why a single step with such codes is much more expensive than with classical methods. We must somehow use just a few evaluations of f to find directions in which f changes rapidly. If we had J available, we could use the power method, Wilkinson [1965], to compute $\rho(J)$ and take it to be a reasonable indication of the size of $\|J\|$. We shall develop an analog of the power method for our nonlinear problem to find directions yielding relatively large lower bounds for L.

In applying the power method to the computation of $\rho(J)$, where J is evaluated at the initial data $x = a$, $y = y(a) = y^{(0)}$, we would form $u, Ju, J^2u, \ldots$. Normally when using the power method, the vectors $J^m u$ are scaled so as to control their growth. After selecting $y^{(1)}$ in a manner to be described later, we define $u = y^{(1)} - y^{(0)}$ and evaluate Ju from

$$f^{(1)} - f^{(0)} = f(a, y^{(1)}) - f(a, y^{(0)}) = J_1(y^{(1)} - y^{(0)}) = J_1 u.$$

Here J_1 is the Jacobian matrix with its entries evaluated on a line between $y^{(1)}$ and $y^{(0)}$. The issue of scaling is very serious in the nonlinear case because we wish to approximate J_1 by J. Essentially we are forming a

difference approximation to a directional derivative, hence we must use a small increment. In general we define $y^{(m+1)}$ by

$$y^{(m+1)} - y^{(0)} = \frac{1}{\rho_m}(f^{(m)} - f^{(0)}),$$

where ρ_m is chosen to keep the increment small. In fact it is convenient to keep it constant by defining

$$\rho_m = \frac{\|f^{(m)} - f^{(0)}\|}{\|y^{(m)} - y^{(0)}\|} \quad \text{for } m = 1, 2, \ldots.$$

Notice that $\rho_m \leq L$ and $\rho_m \leq \|J_m\| \doteq \|J\|$. Now we find

$$\begin{aligned} y^{(m+1)} - y^{(0)} &= \frac{1}{\rho_m} J_m (y^{(m)} - y^{(0)}) \\ &= \frac{1}{\rho_m \rho_{m-1} \cdots \rho_1} J_m J_{m-1} \cdots J_1 (y^{(1)} - y^{(0)}) \\ &\doteq \frac{1}{\rho_m \rho_{m-1} \cdots \rho_1} J^m (y^{(1)} - y^{(0)}). \end{aligned}$$

This is like the power method applied to J with scaling playing both the usual role and that of maintaining a reasonable linear approximation to f. A different and fruitful way of thinking about the process is that we are examining directional derivatives from $y^{(0)}$ and that we rotate the direction so as to find one of large change.

Because we are interested in finding a direction of large change in f, we keep track of the largest lower bound ρ_m for L that we encounter. An expansion of $y^{(m)} - y^{(0)}$ in terms of eigenvectors of J shows that successive iterates are enriched in the directions corresponding to large eigenvalues. For our purposes, seeing the presence of large eigenvalues suffices and we have no need to isolate the largest. If the largest eigenvalue is real and simple, the vector $y^{(m)} - y^{(0)}$ converges to the corresponding eigenvector. In such a case our scheme will find a bound about the size of the spectral radius, although it might fortuitously discover a much larger value. A more interesting possibility is that the largest eigenvalues are complex conjugates. Then the vector does not converge; it moves about in a subspace corresponding to the associated eigenvectors. If the dominant eigenvalues are $\lambda = r\exp(i\vartheta)$ and its conjugate, then the jth component of the iterate behaves like

$$\frac{(J^{m+1}u)_j}{(J^m u)_j} \sim r\frac{\cos(\varphi + m\vartheta + \vartheta)}{\cos(\varphi + m\vartheta)},$$

where φ depends on u. As this observation suggests, our lower bound can fluctuate between values much smaller than $r = \rho(J)$ and much larger. Of course we could calculate an approximate spectral radius if we wished by using techniques developed for this case. We do not do so because it is a large lower bound for L we want, not the spectral radius.

In our subroutine for approximating L we have limited ourselves to three iterations. In our experience this is ordinarily more than enough to reveal the presence of large eigenvalues. Only three evaluations of f are made which are not used in the direct solution of the differential equation. This is a modest cost for the safety and information gained.

After we wrote our subroutine we found that Lindberg [1973] had done something rather similar in his IMPEX 2 code. We were surprised to discover this because his code is intended for stiff problems; we do not understand why he did not simply compute a norm of the Jacobian. The differences betwen the subroutines are interesting and serve to point out some of the care needed.

So far we have not specified $y^{(1)}$. Lindberg makes an artificial perturbation of the initial point $y(a)$. He uses it for the role played by our $y^{(0)}$ and uses $y^{(0)}$ for the role played by our $y^{(1)}$. Rather than an artificial perturbation, we have chosen $y^{(1)}$ by

$$y^{(1)} - y^{(0)} = \frac{1}{\rho_0} f(a, y^{(0)}),$$

where ρ_0 is used to normalize the difference. We hope to select a vector $y^{(1)} - y^{(0)}$ which is at least not extremely deficient in the directions corresponding to large eigenvalues. From the differential equation, $y'(a) = f(a, y(a))$, so if the initial slope is large, it will be reflected in our choice and the initial scale revealed. Approximating

$$f(a, y^{(0)}) \doteq Jy^{(0)} + g,$$

we can interpret the choice as an amplification of the initial vector $y^{(0)}$ in the directions corresponding to large eigenvalues. These heuristics suggest that our choice of $y^{(1)}$ is not likely to be bad and may even be excellent. The derivative $y'(a)$ will surely be needed for the solution of the differential equation, hence our choice involves no extra expense.

Lindberg keeps the size of the difference $y^{(m)} - y^{(0)}$ constant as we do, but the size chosen is quite different. He first perturbs components of $y^{(0)}$ by a relative change of the tolerance ε if the component is nonzero and by an absolute change of ε otherwise. This vector with no zero components is

the base point for his iteration and the size of the increment is fixed at a relative change of ε in its norm. We think the perturbation should depend on the machine precision, so we fix the size of the change at $\sqrt{u}\|y^{(0)}\|$, where u is the unit of roundoff. This roughly amounts to changing the last half of the digits available. If this size underflows, we make an absolute perturbation of the smaller of $\sqrt{u}$ and $\frac{1}{2}\varepsilon$.

It is certainly possible that $\|f^{(m)} - f^{(0)}\| = 0$ because, e.g., f might not even depend on y. This quantity is used in the normalization of $y^{(m+1)}$ and one can get a divide check in his program if he is not careful. We found this out the hard way and as we read Lindberg's program, it could blow up for this reason. A more critical consequence is that $y^{(m+1)}$ is not defined. In our program perturbations are made along the coordinate axes successively whenever needed for this reason so as to carry out the required number of iterations.

An important distinction is that Lindberg does a fixed number of iterations and accepts the last ratio computed. He describes his subroutine as computing a Lipschitz constant, but it appears that he expected to approximate the spectral radius by the last iterate. As the situation with complex dominant eigenvalues makes clear, this last ratio might be a lot smaller, or larger, than the spectral radius. To approximate a local Lipschitz constant one must use the largest lower bound encountered.

Any vector norm might be used in our scheme, but the (weighted) Euclidean norm has an advantage. In the Euclidean norm, $\|J_m\|^2$ is the largest eigenvalue σ^2 of the symmetric matrix $J_m^{\mathrm{T}}J_m$. A simple manipulation shows that the ratio ρ_m^2 is the Rayleigh quotient for the approximation of σ^2 using the vector $y^{(m)} - y^{(0)}$. It is well known how effective the Rayleigh quotient is for taking advantage of vectors which are poor approximations to a desired eigenvector. Even when the power method is not converging well (or at all) to $\rho(J)$, we are still likely to turn up a reasonable value for $\|J\|_2$.

In our numerical experiments we have used a weighted Euclidean norm. In part this is because the code ODE which we modified is based on this norm and automatically computes a quantity we shall need for our treatment of subsequent steps of the integration.

19.4.2. General step

Monitoring the behavior of the problem at each step will be a significant amount of work unless we keep the cost per step to an absolute minimum.

For this reason we do not propose making any extra function evaluations. Of course, one might wish to provide for a closer examination of the problem on some indication of trouble.

It is easy to get a reasonable lower bound for the Lipschitz constant in an Adams code. All such codes predict a solution value p_{n+1} for the step to x_{n+1}. They then evaluate $f_{n+1}^p = f(x_{n+1}, p_{n+1})$ and correct the solution approximation by forming y_{n+1}. At this point codes proceed differently. One popular procedure, represented by the code ODE, is to accept y_{n+1}, evaluate $f_{n+1} = f(x_{n+1}, y_{n+1})$ and go on to the next step. The other popular procedure, represented by Gear's DIFSUB, continues the correction process until it 'converges'. Descriptions vary, but all the codes estimate the local error by comparing the predicted and corrected values, hence form a multiple of $\|y_{n+1} - p_{n+1}\|$. In a code like ODE one need merely form the norm $\|f_{n+1} - f_{n+1}^p\|$ and compute the ratio to get a lower bound for the Lipschitz constant. It is even easier in a code like DIFSUB which computes this second quantity for use in its convergence test. In either case a cheap bound is readily available.

There is a practical difficulty. It can happen by accident, especially with very stringent accuracy requirements, that y_{n+1} is so close to p_{n+1} that the difference $f_{n+1} - f_{n+1}^p$ is merely the result of roundoff. In our experiments with ODE we did not form the bound unless $\|y_{n+1} - p_{n+1}\| \geq 100u\|y_{n+1}\|$. If the right-hand side were zero, we used instead the larger of $\sqrt{u}$ and half the tolerance parameter ε.

The Adams codes which do repeated corrections present an opportunity to form several approximations. Each iteration amplifies the components of $y_{n+1} - p_{n+1}$ in the directions corresponding to large eigenvalues which should lead to a good value for L. Unfortunately the step size is adjusted so that convergence is very rapid; one must be careful to check that the increment is not too small for the precision if he is to obtain a reliable approximation of L.

The explicit Runge–Kutta methods of s stages have the form:

$$y^{(1)} = y_n,\ f^{(1)} = f(x_n, y^{(1)}),$$

$$y^{(i)} = y_n + h\sum_{j=1}^{i-1} b_{ij}f^{(j)}, \quad i = 2, \ldots, s,$$

$$f^{(i)} = f(x_n + a_ih, y^{(i)}), \quad i = 2, \ldots, s,$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} c_if^{(i)}.$$

The constants a_i, b_{ij}, c_i define the method. In this generality it is not obvious how one can obtain lower bounds for the local Lipschitz constant as we did with Adams methods. The difficulty is that we wish to form a directional derivative in the dependent variables, but the function evaluations at our disposal are at values $x_n + a_i h$ of the independent variable which are, in general, different.

A number of popular methods, e.g., England's, evaluate the function more than once at the same value of the independent variable. The possibility even arises of computing several lower bounds in the course of a step. England's formula has $a_1 = a_2$, $a_3 = a_4$, $a_5 = a_6$, $a_7 = a_8$ which means we can get at least four lower bounds. His formula is based on the classical Runge–Kutta formula, which brings up an interesting historical note. In our notation this formula has $a_1 = a_2$ hence we get a lower bound for the Lipschitz constant from

$$\frac{\|f^{(2)} - f^{(1)}\|}{\|y^{(2)} - y^{(1)}\|}.$$

This formula comes without a local error estimator, so in the early days of computing attention was given to the question of selecting a reasonable step size. Collatz [1970, p. 71] suggested a rule of thumb for the case of a single equation. His description is a bit ambiguous, but tracing through a couple of cross references and a paper in the literature shows that he intended that the product of the step size and the bound just stated be kept of modest size. This is a perfectly appropriate necessary condition. Because it does not yield an h small enough to produce the desired accuracy, its usefulness was not appreciated later when satisfactory local error estimators became generally available.

It is very common, one might say typical of high order formulas, that at least one of the evaluations be at $x_n + a_i h = x_{n+1}$. It is convenient to program a Runge–Kutta method so that on a successful step one evaluates $f(x_{n+1}, y_{n+1})$ along with the formation of y_{n+1}. This represents the first evaluation for the next step. By programming the computation so as to overlap a little with the next step, one gets two function evaluations with $x = x_{n+1}$. England's formula permits this device too.

One of these two possibilities is applicable to all the formulas considered in the study of Shampine and Watts [1977]. There is also an approach which can always be used. If one were to put (19.1) in autonomous form by introducing a new independent variable t such that

$$\frac{dx}{dt} = 1, \quad x(a) = a,$$

the special role played by x would disappear and all of the function evaluations would be available for forming difference quotients. Mathematically the two formulations of the problem are equivalent, but computationally they are not. The implications are a little subtle, and we refer the reader to the discussion of Shampine [1978]. In the present context we note that the effect on the eigenvalues of the Jacobian of introducing the new variable t is to add an eigenvalue which is zero. The effect on the norm of the Jacobian is to include the variation of the function with respect to x, by no means a bad idea as far as seeing how the scale of the problem goes. We also note that it is not necessary actually to carry out the integration with the autonomous form; one merely needs to defined properly the vectors used in the difference quotient.

The idea for approximating the local Lipschitz constant proposed in this section works surprisingly well. Our use of a Rayleigh quotient, as described in the last section, takes advantage of whatever vectors are available, but why are directions corresponding to large eigenvalues so well represented in $y_{n+1} - p_{n+1}$? We do not completely understand this. One aspect is that the stability regions of the predictors are much smaller than those of the corresponding correctors. The computation can be proceeding stably while the prediction itself is unstable — meaning that directions corresponding to large eigenvalues are amplified in p_{n+1} but not in y_{n+1}. A closely related observation is that if we approximate f by $Jy + g$, then the predictor has the form $p_{n+1} = h\alpha J y_n + \Psi_n$, where α is a constant defined by the formula. This approximation shows the amplification in the directions corresponding to large eigenvalues. The situation is not essentially different with Runge–Kutta methods because they have the same structure of prediction and evaluation. Van der Houwen [1977] considers the 'internal stability' of Runge–Kutta formulas which amounts here to examining the stability of the 'predicted' values.

19.5. Detecting stiffness

The methods proposed in Section 19.4 are rather effective for detecting large Lipschitz constants. A problem with a large Lipschitz constant is locally either very stable or unstable. Most problems to be solved numerically are well posed, which is to say, they are stable. For this reason a large

Lipschitz constant is prima facie evidence of a stiff problem. In this section we consider what additional evidence is readily available to decide whether the problem is stiff.

In a series of works, see, e.g., [1975, 1977], Shampine has investigated the behavior of numerical methods like Adams and Runge–Kutta when confronted with a stiff problem. Briefly, what happens is that when working with a step size such that the method is stable, propagated errors are damped and the error estimator eventually realizes that the smooth solution being tracked permits a much larger step size. When the step size is increased to the point that it corresponds to being outside the stability region of the method, propagated errors grow and the error estimator eventually sees that the step size must be reduced. Thus the step size used by the code will correspond, on the average, to being on the boundary of the stability region. Stiffness means that the desired accuracy could be achieved with a step size much larger than that necessary for a stable computation.

There is a detail about the step size that must be handled. For a variety of algorithmic reasons a code will select a step size smaller than appears will succeed. Also, a variable order code like ODE may change the order before estimating the step size for the next step. At least a few lines of code must be added to the typical differential equation solver to obtain the step size which is estimated will succeed on the next step if taken at the current order. This is an easy modification to the program.

We want to know how much the step size must be restricted for the computation to be stable. Let c be the radius of the largest disc contained in the stability region of the method. If $h\rho(J) > c$, the computation can be unstable. From this we find how much the step size must be reduced to be sure of stability. The stability regions are not discs so the condition to obtain stability may be rather severe. If the Lipschitz constant is large and if the step size must be reduced by at least a factor of 10 to assure stability, we shall say the problem is stiff.

The spectral radius $\rho(J)$ is not available to us for the test proposed. Our algorithms produce an approximation to the local Lipschitz constant which is quite likely to lie between $\rho(J)$ and $\|J\|$. If this is the case, the test is valid with the approximation; the test just may be quite conservative. If the approximation is smaller than $\rho(J)$, the test is not valid with the approximation and a mistake is possible. Unless the approximation is much smaller that $\rho(J)$, the conservative nature of the test is likely to prevent wrongly calling the problem stiff. It is possible that a mistaken diagnosis of stiffness

is made. For this reason we emphasize the rigorous statement that the Lipschitz constant is large and only add the statement that the problem appears to be stiff. A mistake is really very unlikely. In the first place, the fact that the Lipschitz constant is large and the very high probability that the problem is well posed, make it very likely that the problem is stiff without further investigation. The algorithms we have proposed are likely to produce an approximate Lipschitz constant of a size at least comparable to the spectral radius. The conservative approach to measuring the restriction provides substantial protection against a mistaken diagnosis of stiffness. The experimental results we present in Section 19.6 agree that we are being extremely conservative about calling a problem stiff.

19.6. Numerical experience

We have modified the code ODE as described in the preceding sections and submitted it to a great many tests. Here we shall report some of the results and discuss in some detail the more illuminating experiments. The code already has a test for stiffness, Shampine and Gordon [1975, Ch. 8], to which the developments of this paper will be compared.

There are a couple of additional details about our modification of ODE that need comment. When the code is doing its initialization, it forms an approximation BNDA of the Lipschitz constant in a subroutine as described in Section 19.4.1. It has a procedure for automatically selecting the initial step size h. We modified the code so that this h is reduced as necessary to make $h * \text{BNDA} \leq 1$. No restriction of this kind is imposed on the step size after the first step. At the end of each *successful* step the procedure described in Section 19.4.2 is applied to form a new approximation BND to the local Lipschitz constant. On the first step the larger of BNDA and BND is used and thereafter only BND is used. The *current* value of BND is used to decide if the Lipschitz constant is large and it is the *remaining* interval of integration that is relevant.

19.6.1. Non-stiff problems

The performance of the algorithms on the set of non-stiff test problems assembled by Hull et al. [1972] is not without interest. The set of 25 test problems was solved for the representative pure absolute error tolerances 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}. The problems called D1–D5 are a family of two body problems with one parameter. The orbits become more eccentric as one

goes from D1 to D5 and the integrations become much harder. As it turns out, it is quite easy to obtain analytically the eigenvalues of the Jacobian J and $\|J\|_2$ at the initial data for D5. The eigenvalues there are $\pm\sqrt{2000}$ and $\pm i\sqrt{1000}$ and $\|J\|_2 = 2000$. Numerical computation of the eigenvalues at the initial data show a similar pattern for other members of the family with the magnitudes increasing as one goes from D1 to D5. By our definition D5 has a large local Lipschitz constant at the initial point with the given interval [0, 20]. This is correctly reported by the code. In fact, a large constant is reported for several steps in the neighborhood of the initial point. Naturally the number of steps depends on the tolerance. For tolerances 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8} the code reported a large local Lipschitz constant 2, 3, 3, 7 times, respectively.

With the exception of D5 no problem in the test set was diagnosed as having a large local Lipschitz constant. D5 was not diagnosed as being stiff, which is the correct action.

19.6.2. Stiff problems

To understand the performance of our algorithms on the set of stiff test problems assembled by Enright et al. [1975], it is first necessary to discuss some of the problems. The problem E2 is the solution of van der Pol's equation on [0, 1]. We believe the problem as stated in the literature is incorrect, for the solution and its first derivative change modestly on this interval and the largest eigenvalue of the Jacobian is reported to be only -15. We do not call this a stiff problem. Our practical definition is confirmed by the fact that this problem is an extremely easy one for ODE. The problems B2 and B3 are more debatable. They are posed on [0, 20] and their constant Jacobians have spectral radii of about 10.4 and 12.8, respectively. The set of *non-stiff* test problems of Hull et al. [1972] includes one, C2, posed on [0, 20] which has a constant Jacobian with dominant eigenvalue -9. There is no practical way to distinguish the stiffness characteristics of this problem and of B2. We defined a large Lipschitz constant as being at least 500 when the interval remaining to be integrated is scaled to [0, 1]. By this definition E2, B2, B3 do not have large Lipschitz constants. ODE handles the problems correctly so we need refer no more to them below. Perhaps we should mention that even at the most stringent tolerance in our tests, ODE solved these problems in 57, 276, 392 steps, respectively. The old test for stiffness will not even consider the question until 500 steps have been taken.

All the stiff problems were integrated at the pure absolute error

tolerances 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}. The test for a large local Lipschitz constant is basically independent of the tolerance. At the initial point the tolerance plays a role only if the initial values are all zero, of which several examples occur. On subsequent steps it plays a role only in that at stringent tolerances the predicted and corrected values are more likely to be so close together that a difference quotient cannot be formed reliably.

At all tolerances the code reports that a large local Lipschitz constant is present at the initial point for all problems except E1. (Remember some problems were dropped from the set as not being stiff.) The issue is not quite as straightforward as this appears. Of course we need to look at E1, but another problem needs to be examined as well.

The problem D2 (Robertson's example) does not have large eigenvalues at the initial data. They are 0, 0, -0.04. According to these eigenvalues this problem is not stiff initially. (The eigenvalues increase rapidly as the integration proceeds.) However, at the initial data $\|J\|_2 \doteq 400$, showing clearly the difference between $\rho(J)$ and $\|J\|$. Our algorithms behave in an interesting way. The iterative procedure applied at initialization computes $\rho(J)$ accurately without encountering a direction in which the size of $\|J\|$ is revealed. In the first step there are two tests. The one at the end of the step shows the presence of a large Lipschitz constant (and the increased $\rho(J)$).

The situation with E1 is also illuminating. The spectral radius $\rho(J) \doteq 147$ throughout the interval [0, 1]. If it were the case that $\rho(J) \doteq \|J\|$, the Lipschitz constant would not be large enough for us to classify it as 'big'. It is easy to see that $\|J\|$ is in fact quite large. The general result

$$\|J\|_2 \geq \frac{\|Je\|_2}{\|e\|_2} = \left(\sum_{i=1}^{n} J_{ij}^2\right)^{1/2} \geq |J_{ij}|,$$

where e is a vector with 1 in its jth position and 0 elsewhere, and the fact that $J_{41} = -10^8$ at the initial data show the size of $\|J\|$. The following argument has more the flavor of the scheme we use and shows why one must worry about large $\|J\|$. The first three equations do not depend on the first solution component y_1 and the last equation is

$$y_4' = (y_1^2 - \sin(y_1) - \Gamma^4)y_1 + \left(\frac{y_2 y_3}{y_1^2+1} - 4\Gamma^3\right) y_2 + (1 - 6\Gamma^2)y_3 + (10\exp(-y_4^2) - 4\Gamma)y_4 + 1.$$

The initial vector is 0. Suppose the first component $y_1 = 0$ is perturbed to δ. Then the lower bound (19.6) is

$$\frac{\Gamma^4\delta + O(\delta^3)}{\delta} = \Gamma^4 + O(\delta^2).$$

Because $\Gamma = 10^2$, we have exhibited a direction in which f is very sensitive to its arguments. Our automatic scheme is not this successful in the first step. It computes a lower bound of about 390 which is considerably bigger than $\rho(J)$, but still not 'big' by our definition. Thus the code does not report a large Lipschitz constant on the first step. It does, however, detect a large value before going far and then needs only a few steps to decide the problem is stiff.

Normally there is an initial transient which requires a small step size for its resolution. Although the local Lipschitz constant is large and the problem is stable, this phase of the integration should not be described as stiff. The tolerance obviously plays an important part here because the more stringent the tolerance, the more steps will be needed to resolve the transient. In particular, for a typical problem the more stringent the tolerance, the more steps ODE should take before reporting stiffness. A number of the test problems have initial values specified near a smooth solution and exhibit little, if any, initial transient. In such a situation ODE might well report the integration stiff on the first step. This happened for

tolerance	problem
10^{-2}	D1, D4, D6, E5
10^{-4}	D1, D4, D6, E5
10^{-6}	D6, E5
10^{-8}	E5

Reminding the reader that problems B2, B3, and E2 were excluded as not having large Lipschitz constants, we shall try to give some idea of the results for the remaining problems. The difficulty in getting a fair appreciation of the code's success is that the results for one or two problems distort the averages.

At the tolerance 10^{-2} the code reported the problem stiff at the 29th step, on the average. The number of steps was 170 for B5 but the unusual behavior for this problem is made clear by the fact that the maximum for the other problems was 84. An important point is that at practically every step the code reports that the local Lipschitz constant is large until it diagnoses stiffness and we terminated the run. For example, this happened at every step with B5. Repeated messages to the effect that the Lipschitz constant is large should alarm the user. If he is reasonably confident that his problem is well posed, he should presume the problem stiff. If not, he should examine his equations for correctness and his model for its validity.

We have pointed out that in some contexts one tries to deal with unstable problems, but such problems are hardly typical.

The results at all the tolerances are similar. At 10^{-4} an average of 58 steps was required to detect stiffness. B5 took 288 steps and the maximum over the other problems was 125. At 10^{-6} an average of 92 steps was required to detect stiffness. B5 took 469 steps and the maximum over the rest was 155. At 10^{-8} an average of 205 steps was required to detect stiffness. The problem E4 required 1097, B5 required 720, A4 required 564 and none of the rest required more that 277.

Something special happened with B4 at the tolerances 10^{-4}, 10^{-6}, and 10^{-8}. At 10^{-4} the code reported that the Lipschitz constant was large 85 times but carried out the entire integration in 649 steps without diagnosing stiffness. At 10^{-6} the code reported a large Lipschitz constant 105 times but completed the integration in 653 steps without declaring the problem stiff. At 10^{-8} a large Lipschitz constant was reported 154 times and the integration completed in 754 steps without diagnosing stiffness. The constant Jacobian has $\rho(J) \doteq 27$. On the interval [0, 20] the code must detect stiffness pretty quickly or else the interval remaining for the intergration will be too short to call the Lipschitz constant large by our definition. Of course the more stringent the tolerance, the less stiff the problem appears in the first part of the interval. The results for B4 at tolerances 10^{-4}, 10^{-6}, 10^{-8} were not included in averages given for the number of steps needed to detect stiffness.

The new algorithms in ODE are much more effective than the old test for stiffness. On the other hand, the old test is cheap and based on a different principle so we prefer to leave it in the code. Only a few of the integrations reported were allowed to take enough steps (500) that the old test could even by applied. The old test proved helpful with B4 because it diagnosed stiffness when 500 steps was reached for each of the tolerances 10^{-4}, 10^{-6}, 10^{-8}. The only other integrations requiring more than 500 steps were at the tolerance 10^{-8} for which A4 required 564 steps, B5 required 720, and E4, 1097. The old test was not helpful in these cases. It did detect stiffness but was less efficient since it needed 1500, 1000, and 2500 steps, respectively. It is worth emphasizing that the new test is returning an alarming number of messages that the local Lipschitz constant is large. Specifically, there were 561 such returns when solving A4 before stiffness was diagnosed, 719 returns when solving B5, and 1086 when solving E4. This is to say that at virtually every step the code warned us that something was wrong; it just took a lot of steps to be more specific about the cause.

To close this section we mention a few results for the problem described in Section 19.1 which Scott and Watts solved. The behavior is complicated because it depends on the tolerance. At the representative tolerance 10^{-6} the problem with initial value 0.99026 was integrated in 413 steps. In the course of the integration a Lipschitz constant as large as 3475 was discovered. Because the large constants occur near the end of the integration, they are not reported by the code as large by our definition. This is proper because the integration is not expensive. With initial value 0.99, the code reported a large Lipschitz constant when it reached 0.629. At this point it had taken 206 steps and it had a lower bound for the Lipschitz constant of 1353. On each of the next 54 steps the code reported that the constant was large and finally declared the problem stiff at 0.654. At this point the local Lipschitz constant was 1955. Regardless of whether one chooses to quit on a report of a large Lipschitz constant or of probable stiffness, this is a much better response than is possible with the old test in ODE.

19.7. Conclusions

The methods proposed for monitoring the local Lipschitz constant are simple, inexpensive, and of broad applicability. They enhance the reliability of numerical methods in the difficult regime of nominal accuracies and detect severe difficulties with scaling. The additional device proposed for determining stiffness is crude, but experiment shows it to be useful. The value of the information obtained by our methods seems high compared to the cost of getting it. Anyone constructing a new code or seriously modifying an old code should consider the possiblity of adding these algorithms.

Acknowledgment

The modification of ODE and most of the computations reported in this paper were made by K.W. Stewart. Discussions with her helped the author to straighten out his ideas and substantially influenced the implementation of the algorithms.

References

L. Collatz [1960], The numerical treatment of differential equations, 3rd ed., Springer, Berlin.

W.H. Enright, T.E. Hull and B. Lindberg [1975], Comparing numerical methods for stiff systems of ODEs, BIT 15, pp. 10–48.

C.W. Gear [1971], Numerical initial value problems in ordinary differential equations, Prentice-Hall, Englewood Cliffs, NJ.

P. van der Houwen [1977], Construction of integration formulas for initial value problems, North-Holland, Amsterdam.

T.E. Hull, W.H. Enright, B.M. Fellen and A.E. Sedgwich [1972], Comparing numerical methods for ordinary differential equations, SIAM J. Numer. Anal. 9, pp. 603–637.

T.E. Hull, W.H. Enright and K.R. Jackson [1976], User's guide for DVERK — a subroutine for solving non-stiff ODEs, Dept. of Comp. Sci. Rept. 100, University of Toronto, Toronto, Canada.

K.R. Jackson, W.H. Enright and T.E. Hull [1978], A theoretical criterion for comparing Runge–Kutta formulas, SIAM J. Numer. Anal. 15, pp. 618–641.

B. Lindberg [1973], IMPEX 2 a procedure for solution of systems of stiff differential equations, Dept. of Information Processing Rept. TRITA-NA-7303, Royal Institute of Technology, Stockholm, Sweden.

M.R. Scott and H.A. Watts [1976], A systematized collection of codes for solving two-point boundary-value problems, in: L. Lapidus and W. Schiesser, Eds., Numerical Methods for Differential Systems, Academic Press, New York, pp. 197–227.

L.F. Shampine [1974], Limiting precision in differential equation solvers, Math. Comput. 28, pp. 141–144.

L.E. Shampine [1975], Stiffness and non-stiff differential equation solvers, in: L. Collatz, Ed., Numerische Behandlung von Differentialgeichungen, Birkhauser, Basel, Switzerland, pp. 287–301.

L.F. Shampine [1977], Stiffness and non-stiff differential equation solvers, II: detecting stiffness with Runge–Kutta methods, ACM Trans. Math. Software 3, pp. 44–53.

L.F. Shampine [1978], Limiting precision in differential equation solvers, II: sources of trouble and starting a code, Math. Comput., pp. 1115–1122.

L.F. Shampine and C.W. Gear [1979], A user's view of solving stiff ordinary differential equations, SIAM Review 21, pp. 1–17.

L.F. Shampine and M.K. Gordon [1975], Computer Solution of Ordinary Differential Equations: The Initial Value Problem, Freeman, San Francisco.

L.F. Shampine and H.A. Watts [1977], The art of writing a Runge–Kutta code, part I, in: J.R. Rice, Ed., Mathematical Software III, Academic Press, New York, pp. 257–275.

J.H. Wilkinson [1965], The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.