Journal of Computational and Applied Mathematics 18 (1987) 175–192
North-Holland

Automatic selection of the initial step size for an ODE solver

I. GLADWELL *
Department of Mathematics, University of Manchester, Manchester M13 9PL, United Kingdom; and, Numerical Algorithms Group Ltd., Mayfield House, 256 Banbury Road, Oxford OX2 7DE, United Kingdom

L.F. SHAMPINE *
Numerical Mathematics Division 1642, Sandia National Laboratories, Albuquerque, NM 87185, U.S.A.

R.W. BRANKIN **
Department of Mathematics, University of Manchester, Manchester M13 9PL, United Kingdom

Received 2 October 1985
Revised 24 June 1986

Abstract: The choice of initial step size is critical for the reliable numerical solution of the initial value problem for a system of ordinary differential equations. Automatic selection of this step size may lead to a more robust and efficient integration than its provision by a user, and is always more convenient. It is especially important for the reliability of an ODE solver used as a module in a larger software package.

Previous approaches to making the selection are combined with some new ideas to produce an effective scheme for the automatic choice of the initial step size. Numerical results illustrate the roles played by the individual phases of the algorithm and show that the whole algorithm is both robust and efficient.

Keywords: Initial value problems, step size selection, ordinary differential equations.

1. Introduction

The popular codes for the numerical solution of the initial value problem for a system of ordinary differential equations (ODEs),

$$y' = f(x, y), \quad a \leqslant x \leqslant b, \qquad y(a) \text{ given},$$

proceed by steps from the solution given at $x = a$ to the final point $x = b$. The first step is critical because the justification of the algorithms for the adjustment of the step size on subsequent steps depends on making small changes. In our experience, casual users of ODE solvers do not wish to be involved in specifying an initial step size and are often unable to provide a suitable value. An important use of ODE solvers is as a module in a larger code whefe it is essential that the initial step size be determined in an automatic and highly reliable way.

* Current address: Dept. of Math., Southern Methodist University, Dallas, TX 75275, U.S.A.
** Current address: Numerical Algorithms Group Ltd., Mayfield House, Oxford OX2 7DE, U.K.

There have been a number of attempts to devise algorithms for the automatic selection of initial step size. In the next section we categorise these approaches and provide some references to their historical development. In Section 3 we present an algorithm which combines a number of the approaches with some ideas of our own. Our algorithm is based on a series of observations:

(a) if we can find an initial step size for which the error estimates are valid then we can devise a simple and inexpensive algorithm to improve this step size to a near 'optimal' one;

(b) a very cautious trial initial step may be taken and the information thus generated from within the step used to find a step size for which we can believe the error estimate;

(c) to choose even the trial initial step we require scale information, to find which we have available only the (sometimes inadequate) information from the initial point.

These three observations, in reverse, provide the basis for the three phases of our algorithm. It is difficult to measure the effectiveness of such algorithms, but in Section 4 we attempt this. Our main interest there is in illustrating the roles played by the different phases of our algorithm and in demonstrating that the algorithm is effective and efficient.

2. Previous approaches

In this section we categorise previous approaches to the selection of initial step size. Watts [14] surveys this field from a historical perspective so we need not detail earlier investigations here, but we do amplify his survey in a couple of instances particularly pertinent to our work.

Obviously one possibility for selecting an initial step size is to request the user to supply it. As we shall explain in section 3.4, this is a desirable option even when an effective automatic procedure is available. What concerns us here, though, is the automatic selection of the step size.

Perhaps the simplest scheme used for deducing a step size is to take a fraction of the length of the range of integration or of a maximum step size specified by the user. This is more useful than might be immediately obvious because often the problem statement provides implicitly an indication of scale. A variation, used in DVERK [6], is to multiply a given maximum step size by an appropriate fractional power of the error tolerance. This is designed to provide the behaviour expected on a change of tolerance.

The first non-trivial scheme for automatic selection of the initial step size seems to be that described in [12]. It takes into account the initial slope of the solution as well as the error tolerance and the type of error control. Variations on this theme have been used in many codes since. The basic idea involves approximating $y(x)$ near $x = a$ by the linear terms of Taylor series expansion. It is natural to think of adding more terms to the expansion. The next term involves the Jacobian matrix. In the context of stiff problems using this term is a convenient and effective way to proceed as Curtis [1] explains.

Another approach is based on approximating Lipschitz constants. As employed by Shampine [10] it is quite distinct from the Taylor series approach. However, Lindberg [7] and Watts [14], in effect, use a Lipschitz constant as a substitute for the Jacobian matrix required in the second order Taylor series approach. It is a simple and practical alternative far better suited to the solution of non-stiff problems.

The schemes just mentioned are intended mainly to produce a step size which will succeed. It is hoped that the error estimate will be reliable, so that an efficient step size can be found later in

the integration. Following discussions with Stetter (see [13] for a summary of his views), Gladwell [4] sought to determine the locally optimal step size before starting the integration. (The locally optimal step size is the smallest step size for which the norm of the local error estimate equals the local error tolerance.) Finding this step size is a task which follows naturally one of the schemes for finding a successful step size. Sedgwick [8] does not clearly separate out the starting phase in his code, but, in effect, he starts with the smallest meaningful step size (relative to unit roundoff) and subsequently increases it to a locally optimal value.

3. Initial step size algorithm

We present an algorithm which combines the best features of earlier algorithm for the selection of the initial step size together with some new ideas of our own. It is perhaps best to describe our approach as a methodology because in contrast to the approach taken by Watts, ours is best implemented by modification of the integrator. We have in mind that the problem is non-stiff over the first step. This does not mean that we cannot treat stiff problems with, say, a BDF code because for stiff problems usually it is thought prudent to take the first step small enough to resolve initial transients.

Ideally the initial step size algorithm would produce cheaply and reliably the most efficient initial step size, the 'locally optimal' one. All modern codes estimate the local error of each step. If the error is too large, the step is rejected and tried again with an appropriately reduced step size. If the error is too small, a more efficient larger step size is predicted for the next step. In view of this, a natural way of dealing with the initial step is to supply a crude guess somehow and then rely upon the usual algorithms. This works fairly well, but there are two main difficulties. The dangerous one is that when presented with a step size which is too large the usual local error estimation algorithm may fail, and a step might be accepted when it should be rejected. The other is that the usual step size adjustment must be cautious, with the consequence that a poor guess might require a substantial number of steps to be increased to an appropriate value. The first two phases of our algorithm address the first difficulty. Their aim is to produce a step size for which the local error estimate is credible. At the same time, the step size should not be far too small. The third phase adjusts this step size to a locally optimal value. It is based on the usual algorithm, but is more efficient because it takes account of the special circumstances.

In Phase 1 only data available at the initial point of the integration is used to predict a step size H. The procedure is a variation of a widely used scheme. The code does not have enough information at its disposal to produce a step size tailored to the integration method. Nevertheless, experience over many years and with many codes tells us that the step size produced, H, is almost certainly not disastrously bad, and is often 'on scale'. (An initial step size is 'on scale' if the initial step is successful and the predicted step size increase is within that permitted on a normal integration step.) This phase is nearly free.

In Phase 2 we actually try a step with the integration method. The computation is monitored carefully and the step size is reduced (quickly and inexpensively) as necessary to obtain a credible error estimate. If the result H of Phase 1 is not too large, Phase 2 costs no extra function evaluations and does not reduce H. This new algorithm, which applies ideas in [10], appears to be an inexpensive and effective way to monitor the initial step so as to gain confidence in the error estimate.

Experience with variations of Phases 1 and 2 in other codes and experience which we report in Section 4 shows that often no adjustment of step size is needed in Phase 3. When adjustment is necessary, it is accomplished inexpensively. In particular, when Phase 3 is entered with a step size which is too small, adjustment of the initial step size in this phase is cheaper than starting the integration and letting the usual step size algorithm gradually bring it up to an appropriate value.

3.0. Preliminaries

In his specification of a problem, the user provides implicitly some information about the scale of the problem. This may be done with a maximum step size, a guessed initial step size, or a first output point. Codes use such information to place limits on the initial step size. However this is done, we suppose in the following that the first step proceeds from the initial point a towards a suitably defined point b, but it is not to stretch past the point b.

The nature of the local error control affects the selection of the step size. We suppose that on each step it is required that

$$\| \text{local error} \| \leqslant \tau \tag{3.1}$$

for a scalar tolerance τ supplied by the user. (This is for error per step control; error per unit step is handled similarly.) We take the norm to be a weighted maximum norm where the local error of the approximation to the component $y_i(x_n + h)$ is measured relative to the weight

$$w_i = \max(\text{THRES}_i, \ |y_{n,i}|, \ |y_{n+1,i}|). \tag{3.2}$$

Here $y_{n,i} \simeq y_i(x_n)$, $y_{n+1,i} \simeq y_i(x_n + h)$, and THRES is a vector of threshold values, various choices of which permit both relative and absolute error control. This is a robust scheme which copes well enough with solution components which vanish, except possibly those which vanish at the initial point $x = a$. In the following subsections we must give some attention to the issue of choosing initial weights w_i. For simplicity we prefer to require that if $y_i(a) = 0$, then the user must specify $\text{THRES}_i > 0$. It is easy for a user to recognize and respond to this requirement. It is a reasonable requirement because $\text{THRES}_i = 0$ in this case corresponds to requesting a pure relative error control on the ith solution component, and the user cannot expect to use this type of control when the component vanishes anywhere, and so specifically not when it vanishes at the initial point. The presentation of our approach will be couched in terms of an explicit Runge–Kutta formula. However, it will be clear immediately that other methods for the solution of non-stiff problems may be accommodated in the same methodology.

The precision available on the computer being used imposes some restrictions which are pointed out in [9]. Because the error control postulated is of a relative type, it is sensible, and possibly necessary, to insist that τ be no smaller than a small multiple of *uround*, the smallest positive number such that $1 + \text{uround} > 1$ in the arithmetic of the computer. Details do not matter here, but we do suppose in the following that $\tau > \text{uround}$.

3.1. Phase 1

In this phase we hope to determine the general size of a step appropriate to a one-step method before actually trying such a step. One of the ideas exposed in the first non-trivial scheme [12] for the selection of the initial step size, and subsequently developed further by a number of authors,

is to estimate the local error of a Taylor series formula. The formula of order m has

$$y(a+h) \simeq y(a) + hy'(a) + \frac{h^2}{2!}y''(a) + \cdots + \frac{h^m}{m!}y^{(m)}(a).$$

The local error of this formula is asymptotically

$$E_m = \left\| \frac{h^{m+1}}{(m+1)!} y^{(m+1)}(a) \right\|. \tag{3.3}$$

The other key idea is to postulate that the norm of the local error of *any* one-step method of order $p \geqslant m$ can be approximated by $E_m^{(p+1)/(m+1)}$. In this approximation the largest step size H which leads to a local error no larger than the given tolerance τ is given by

$$|H| = \left[(m+1)!/\| y^{(m+1)}(a) \|\right]^{1/(m+1)} \tau^{1/(p+1)}.$$

(The rule of thumb proposed in [12] is a little different; this one arose in discussions with H.J. Stetter.)

The defects of this procedure leap to the eye. If $y^{(m+1)}(a) = 0$, or is very small, the local error of the Taylor series formula either does not have the form (3.3) or is not well represented by it. This shows up as a very large, or even infinite, step size H in the formula. All the information supplied about y holds at $x = a$ and so may not be helpful even as far away as $x = a + H$, and hence not helpful when we actually try a step. In particular, the weights of the norm depend on the result of the step, and can only be approximated from information available at $x = a$. The only information supplied about the integration method is its order p, and even this assumes f is smooth enough that the usual order holds. Obviously, this lack of information precludes finding a step size tailored to the formula. The rule of thumb connecting the local error of a *specific* formula of order m to *any* formula of order p can, at best, be useful only in a semi-quantitative way. The procedure does have some virtues. It produces an H which changes properly when the tolerance τ is changed and when independent or dependent variables are scaled.

As in [12] we propose to take $m = 0$. Then, using

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

and the restriction that we do not step past b, we find

$$|H| := \min\left(|b-a|, \tau^{1/(p+1)} \| f(a, y(a)) \|^{-1}\right). \tag{3.4}$$

Because any explicit Runge–Kutta method requires $f(a, y(a))$ the prediction (3.4) is nearly free.

The alternative, $m = 1$, is appealing because it involves more information about $y(a)$ near $x = a$. Now

$$y''(a) = f_x(a, y(a)) + f_y(a, y(a)) f(a, y(a)).$$

The circumstances considered by Curtis [1] are especially favourable. Because he is solving stiff problems, he must approximate the partial derivatives here anyway. When solving non-stiff problems as we do, it is not reasonable to ask a user to supply the Jacobian $J = f_y(a, y(a))$, nor it is practical to approximate J. Watts [14] handles this matter in the following way. He bounds E_1 using

$$\| y''(a) \| \leqslant \| f_x(a, y(a)) \| + \| J \| \, \| f(a, y(a)) \|,$$

and then tries to approximate $\| f_x(a, y(a)) \|$ and $\| J \|$ by numerical differentiation. This is easy enough for the first quantity, but it is necessary to approximate $\| J \|$ with a very few

differences, and it is not clear that this even possible. In [7,10,14] it is explained how, using only a few evaluations of f, a lower bound for $\|J\|$ can be computed, which is quite often a reasonable indication of the size of $\|J\|$. Watts' subroutine HSTART has proved quite successful at predicting a reasonable initial step size. But is the extra trouble and expense of using it as compared with (3.4) worthwhile? Our experience indicates that it is not. HSTART does perform better than (3.4) alone but not when taken together with our Phase 2.

In the formula (3.4) we must be careful with the definition of the weights in the norm because we have not yet attempted the computation of $y_{1,i}$ and so cannot use the usual weights (3.2). We approximate the desired weight by

$$w_i = \max(\text{THRES}_i, \ |y_i(a)|) \tag{3.5}$$

for each i. Our requirement that this quantity be positive prevents a possible breakdown of the computation.

3.2. Phase 2

For the solution of non-stiff problems, it is a fundamental requirement that $|H|L$ be 'not large' where L is a Lipschitz constant for f in a neighbourhood of the solution $y(x)$. This requirement implies that the local solution of the differential equation is stable. It also implies that the usual numerical methods are stable if a a suitable limit on $|H|L$ is chosen. Further implications are discussed in [10]. As we take the step in this phase, we are able to assess $|H|L$ along $y(x)$ and so reduce $|H|$ as necessary to satisfy the fundamental requirement.

The tests of Phase 2 are intended to restrict the step size of Phase 1, if necessary, so as to achieve an H for which the local error estimate of the integrator has some validity. Phase 1 might provide an H which is too large for two reasons. One is that the basic assumptions of Phase 1 might not apply to the problem at hand. Another is that Phase 1 has very limited information at its disposal. In particular, it uses data only at the initial point and takes minimal account of the integration method. In contrast to Phase 1, the tests of Phase 2 are applied to the actual first step of the integration.

In this phase of the algorithm we try cautiously to take a step. The monitoring of a local Lipschitz constant is done in the course of evaluating the formula, so in contrast to HSTART is best done inside the integrator. The fundamental idea is particularly easy to exploit with implicit formulas since their evaluation by simple (functional) iteration produces an estimate of $|H|L$ as a byproduct. For the explicit Runge–Kutta methods which most concern us here the matter is rather more complicated.

An explicit Runge–Kutta method of s stages when applied to the system

$$y' = f(x, y), \quad y_0 = y(a)$$

with step size H has the form

$$u_i = y_0 + H\sum_{j=0}^{i-1} \beta_{i,j} f_j, \quad i = 1, \ldots, s,$$

$$f_i = f(a + \alpha_i H, u_i), \quad i = 0, \ldots, s-1,$$

with $u_0 = y_0$, $\alpha_0 = 0$. Then $y(a+H) \simeq y_1 = u_s$. If some $\alpha_i = \alpha_j$, a lower bound for L can be deduced from

$$\|f_i - f_j\| = \|f(a + \alpha_i H, u_i) - f(a + \alpha_j H, u_j)\| \leqslant L\|u_i - u_j\|.$$

Although this is a useful observation, most Runge–Kutta formulas have few (if any) $\alpha_i = \alpha_j$, so that few lower bounds for L can be obtained in this way. This difficulty is overcome by changing to an autonomous form of the differential system.

For the autonomous form of the system we take

$$\mathrm{d}y/\mathrm{d}t = f(x, y), \quad y(a) = y_0, \qquad \mathrm{d}x/\mathrm{d}t = 1, \quad x(a) = a. \tag{3.6}$$

Much of the formal complication in the treatment which follows is due to the fact that the transformation is done in principle only. Thus we need to work out what happens for the transformed system in terms of the given problem. It is well-known, and easily seen, that where the Runge–Kutta method produces u_i and f_i for the non-autonomous form, $(u_i, a + \alpha_i H)^{\mathrm{T}}$ and $(f_i, 1)^{\mathrm{T}}$ are produced for the system (3.6). Then for any i, j

$$\left\| \begin{pmatrix} f_i \\ 1 \end{pmatrix} - \begin{pmatrix} f_j \\ 1 \end{pmatrix} \right\|_a \leqslant \mathscr{L} \left\| \begin{pmatrix} u_i \\ a + \alpha_i H \end{pmatrix} - \begin{pmatrix} u_j \\ a + \alpha_j H \end{pmatrix} \right\|_a . \tag{3.7}$$

Here $\| \cdot \|_a$ is a norm for the vectors of length $(N+1)$ that we define later. It is convenient to introduce the vectors

$$Y = (y, x)^{\mathrm{T}}, \qquad Y_0 = (y_0, a)^{\mathrm{T}}, \qquad F = (f(x, y), 1)^{\mathrm{T}},$$
$$F_i = (f_i, 1)^{\mathrm{T}}, \qquad U_i = (u_i, a + \alpha_i H)^{\mathrm{T}}.$$

With these definitions (3.6), (3.7) become

$$\mathrm{d}Y/\mathrm{d}t = F(Y), \quad Y(a) = Y_0, \tag{3.6a}$$

$$\| F_i - F_j \|_a \leqslant \mathscr{L} \| U_i - U_j \|_a . \tag{3.7a}$$

Generally the Lipschitz constant $\mathscr{L}$ of (3.7) is not the same as L because variation with respect to the independent variable x is considered. This seems desirable because how fast f changes as a function of x is likely to affect the locally optimal step size. Watts [14] specifically addresses this point as he approximates $f_x(a, y(a))$.

The autonomous form provides an easy way to get some idea as to the size of L, but there is a price to pay which arises in the definition of the norm. We suppose that

$$\| V \|_a = \max(\| v \|, \ | v_{N+1}/w_{N+1} |) \tag{3.8}$$

where v is a vector of N components, $V = (v, v_{N+1})$, $\| \cdot \|$ is the usual norm, but with weights depending both on y_0 and the stages u_i, and w_{N+1} is a weight. In Phase 2, we change the weights used in Phase 1 so that on the ith stage we use

$$w_k = \max(\mathrm{THRES}_k, \ | y_k(a) |, \ | u_{1,k} |, \ | u_{2,k} |, \ldots, \ | u_{i,k} |), \quad k = 1, \ldots, N.$$

This is done to make Phase 2 robust and to simulate better the weights used in a normal step. The difficulty lies in selecting a suitable weight w_{N+1}. Because the formula integrates the adjoined equation exactly, it is possible, and perhaps appropriate, to take w_{N+1} 'large'. With the norm (3.8), (3.7) becomes

$$\| f_i - f_j \| \leqslant \mathscr{L} \max(\| u_i - u_j \|, \ | (\alpha_i - \alpha_j) H | / w_{N+1}). \tag{3.9}$$

If $\| u_i - u_j \| \neq 0$ and w_{N+1} is 'large', we have

$$\| f_i - f_j \| / \| u_i - u_j \| \leqslant \mathscr{L} \tag{3.10}$$

as a lower bound. This suggests that the actual value of w_{N+1} is unimportant, and it is. However, to deal with degenerate cases we must specify a value. We shall do this later after we have made its role clearer.

We require that $|H|\mathscr{L} \leqslant c$ for a suitable value of c which is 'not large'. The fact that $|\lambda| \leqslant \mathscr{L}$ for any eigenvalue λ of the Jacobian $f_y(a, y(a))$ suggests a way of choosing c. From this point of view, a value of c comparable to the radius of a disc approximating the absolute stability region of the Runge–Kutta formula employed is reasonable. For the DPS formulas [11] used in the numerical studies of Section 4, we take $c = 2$ for this reason.

The test in our implementation of these ideas is that

$$|H| \, \| F_i - F_0 \|_a \leqslant c \| U_i - Y_0 \|_a, \tag{3.11}$$

which is the result of an obvious manipulation of (3.7a), the requirement that $|H| L \leqslant c$, and the restriction to $j = 0$ in (3.7a). Generally such a test makes sense only when the difference $U_i - Y_0$ is significant, for example only when

$$\| U_i - Y_0 \|_a > 10 \, \text{uround} \max(\| U_i \|_a, \| Y_0 \|_a). \tag{3.12}$$

Strict inequality is needed here to cope with the degenerate case

$$U_i = Y_0 = 0.$$

In detail, supposing the first step has the form of an explicit Runge–Kutta method, we proceed as follows: For each stage, $i = 1, \ldots, s$, condition (3.12) is checked. Whenever the check is passed, (3.11) is tested. If this test fails, we must reduce $|H|$. We are testing a necessary condition, and our goal is to obtain a small enough $|H|$ quickly so that we can proceed to Phase 3 which will correct it. We must relate changes in step size here to those permitted in normal integration steps. For simplicity we assume that the increase, α, in step size from one integration step to the next is bounded by $1 \leqslant \alpha \leqslant r$ and that the algorithm is symmetric in the sense that decreases are limited by $1/r \leqslant \alpha < 1$. (Most codes do not possess this symmetry property; the changes in what follows for an unsymmetric case are simple.) Bearing this assumption in mind, we reduce $|H|$ to

$$|H| := \frac{c}{r} \max\big((\| U_i - Y_0 \|_a / \| F_i - F_0 \|_a), \, r^{-3} |H| \big). \tag{3.13}$$

The quantity $|H|\mathscr{L}$ is to be limited for a Lipschitz constant $\mathscr{L}$ holding in a neighbourhood of the solution $y(x)$. However, if $|H|$ is much too large, the lower bound may be descriptive of a Lipschitz constant for a 'large' region about $y(x)$. Accordingly we might reduce $|H|$ far more than is appropriate when 'close' to $y(x)$. For this reason we do not allow a reduction of more than $cr^{-4}|H|$. On recognizing a step size is too large according to (3.11), it is reduced as in (3.13) and the step tried again. Our experience has been that when the test (3.11) fails, it is almost always at the first stage. When this is the case, each new step size tried in Phase 2 costs just one evaluation of f. Because Phase 1 performs so well, the step size from it is usually the only one tried in Phase 2. We have observed very few cases where as many as five evaluations of f are required for Phases 1 and 2 combined. Accordingly, these two phases compete well in cost with the routine HSTART which requires five evaluations.

It is possible that $U_i = U_0 = Y_0$ for all the stages i. This can come about because the step size from Phase 1 (or a previous iteration of Phase 2) is extremely small, or because y has a zero of high order at the initial point. The fundamental expression (3.8) makes it clear that in this rare

situation, the value of w_{N+1} is critical. Unfortunately we simply do not have the information needed to select a proper value. To get a scale-invariant weight from the information available to us we have little choice but to take $w_{N+1} = b - a$. With this choice, the test (3.11) will be passed for all sufficiently small $|H|$. In effect we provide a threshold which enables us to bypass Phase 2 in extreme circumstances.

With any automatic procedure there is always the possibility that on trying a step size which is too large, the arguments presented to the routine for evaluating $f(x, y)$ lead to an overflow. If the problem is well-defined, the evaluation of f at $(a, y(a))$ in Phase 1 is possible. Thus the first real chance for an overflow occurs in Phase 2 when $f_1 = f(a + \alpha_1 H, u_1)$ is formed. If, following this, the test (3.11) can be performed, it seems extremely unlikely that a subsequent overflow would occur. Any difficulty, then, is all but limited to the first evaluation made using the step size H from Phase 1. An overflow in this way is an extremely remote possibility, but we note that the user can avoid it by a sensible choice of the upper limit on $|H|$ implied by the input value of b. Watts [14] makes the point that the initial step size algorithm should explore the neighbourhood of $y(x)$, and nowhere else, or automatic procedures can easily get into trouble of this kind. Our algorithm does confine itself as closely as possible to a neighbourhood of $y(x)$.

We iterate the scheme described until $|H|$ has been reduced sufficiently that all the stages are evaluated and the test (3.11) is passed for each U_i such that (3.12) holds. A trial step has then been completed, and the local error can be estimated in the usual norm (3.2) of the code. If this error is greater than the local error tolerance, τ, the step must be rejected. We take this to imply that H is not yet on scale, hence that we cannot trust the error estimate. For this reason we are unwilling to use an 'optimal' reduction factor, and instead we merely reduce H to H/r and try again. When we finally get a successful step, we exit Phase 2.

There is a connection between the tests of Phase 2 and the way Lindberg [7] and, particularly, Shampine [10] approximate the spectral radius of $\mathscr{J} = F_Y(Y_0)$. Leaving out many practical details, Shampine's procedure can be described as constructing a sequence of vectors Y^i with $Y^i - Y_0$ a multiple of $F(Y^{i-1}) - F_0$ and observing that

$$H\big(F(Y^i) - F_0\big) \simeq H\mathscr{J}\big(Y^i - Y_0\big) \simeq \rho_i (H\mathscr{J})^i HF_0,$$

where ρ_i is a normalizing factor. Accepting here the approximation which is obtained by linearization, it is seen that this is a power method for the computation of the spectral radius. More specifically, if $\{v_k\}$ is a complete set of eigenvectors of $\mathscr{J}$, and $\{\lambda_k\}$ the corresponding eigenvalues, and if we expand

$$HF_0 = \sum \delta_k v_k,$$

then in the linear approximation,

$$H\big(F(Y^i) - F_0\big) \simeq \rho_i \sum (H\lambda_k)^i \delta_k v_k.$$

Disadvantages of this approach are that the iterates Y^i may not be 'close' to $y(x)$ and that each computation of $F(Y^i)$ represents an extra function evaluation.

For the quantities we compute in Phase 2,

$$H(F_1 - F_0) \simeq H\mathscr{J}(U_1 - Y_0) = \beta_{1,0}(H\mathscr{J})HF_0,$$

and in the linear approximation, we find that the general term $H(F_i - F_0)$ can be written as HF_0

multiplied by a polynomial in $H\mathscr{J}$ with known coefficients. Except in degenerate cases, the highest order term in $H(F_i - F_0)$ is

$$\beta_{i,i-1} \cdots \beta_{1,0}(H\mathscr{J})^i HF_0.$$

It is clear that we could form a linear combination of $H(F_i - F_0), \ldots, H(F_1 - F_0)$ with known coefficients which agrees in the linear approximation with $H(F(Y^i) - F_0)$. Thus, if we wished, we could compute the spectral radius of $\mathscr{J}$ without function evaluations beyond those used to form the U_i and without leaving the neighbourhood of $y(x)$. In Phase 2 it suffices to compute just the $H(F_i - F_0)$ because if there are λ_j for which $|H\lambda_j|$ is 'large', the components of HF_0 in the directions v_j will normally be amplified greatly in the successive $H(F_i - F_0)$. Accordingly we obtain a 'large' lower bound for $|H|\mathscr{L}$ and so reduce $|H|$ appropriately. From this rough analysis we understand why we can expect to recognize 'large' $|H|\mathscr{L}$ and why this is likely to be observed while evaluating the first few stages of the explicit Runge–Kutta formula.

3.3. Phase 3

When we exit Phase 2 we have made a successful step with a step size H which has passed a considerable number of tests on $|H|\mathscr{L}$. These tests lead us to believe that the error estimator has some validity which we exploit by making 'optimal' adjustments to the step size.

Ideally we would determine the locally optimal step size in Phase 3. If a trial initial step succeeds and the predicted next step, H_{pred}, represents an increase by a factor $1 \leqslant \alpha \leqslant r$ as permitted on a normal step, we describe this initial step as 'on scale'. It is a good approximation to the locally optimal step size which is adequate in view of the algorithm for adjusting it on subsequent steps. In such a case we terminate Phase 3 and continue the integration with H_{pred} as the step size to be attempted on the second step.

An increase factor $\alpha > r$ is taken to imply that we are not yet on scale. In such a case we reject the trial step and try again with the bigger step size

$$|H| := \min\big(|H_{\text{pred}}|, r^3|H|, |b-a|\big). \tag{3.14}$$

It is necessary to restrict the increase in $|H|$ because the error estimate used to predict H_{pred} cannot be considered valid on a range far longer than $[a, a+H]$. The factor r^3 represents a compromise between reliability and efficiency.

It is conventional on finding a successful initial step to begin the integration and so let the code find an 'on scale' value as the integration proceeds. The algorithm for an ordinary step allows an increase by a factor of at most r because prediction of a larger increase is not usually to be trusted. In the special situation of the start, we expect occasionally to make large increases. As a consequence it is possible to get on scale much faster proceeding as we do than by relying on the usual step size adjustment algorithms. An intermediate possibility is common in codes for stiff problems where the usual step size algorithm is modified to something like the following: A step size increase of r^3 is allowed on the first step, r^2 on the second and r on the third and all subsequent steps.

If a trial initial step is a failure, that is, $\alpha < 1$, we reject it as we would when trying a normal integration step. However, we believe that all the checks we have performed are sufficient to

establish the validity of the error estimate, hence that we can take an optimal step size reduction with the aim of finding an on scale step. We choose to restrict H_{pred} as follows:

$$|H| := \max\big(|H_{\text{pred}}|,\ r^{-2}|H|\big). \tag{3.15}$$

Note that $|H_{\text{pred}}| \ll |H|$ implies that a previous error estimate was not reliable, hence that the current one may also be unreliable. In such a circumstance some caution in making a large step size reduction is required. Again the restriction to a lower bound of $r^{-2}|H|$ is a compromise between reliability and efficiency with an eye towards avoiding cycling; a choice of $r^{-3}|H|$ would have made cycling a distinct possibility.

Our iteration in Phase 3 continues until the predicted step size increase α lies in the range $1 \leqslant \alpha \leqslant r$. Of course we are not guaranteed that this iteration will converge. All our numerical tests indicate that not only does it converge, but that convergence is rapid.

A novelty in our approach to calculating an on scale initial step size is the combination of the three phases which, in favourable circumstances, requires no function evaluations except those needed in the integration. An approach taken by Sedgwick [8] illuminates ours. In his variable order, variable step size Adams code VOAS he also seeks a step size which will yield a reliable error estimate and follows it with the usual step size adjustment on subsequent steps to get on scale. His way of obtaining a reliable error estimate is to try a step size about as small as possible in the precision available. The difficulty is that this is an extremely small step size, and increasing it to an on scale value is expensive. To reduce the expense to a tolerable level, Sedgwick allows much larger step size increases than is usually thought prudent. In addition he allows rapid changes of order so that when an on scale step size is found, it will be associated with a suitable order to proceed with the integration. It seems us that the robustness achieved initially by choosing a very small step size to ensure an asymptotically correct local error estimate is likely to be lost on the following steps by trusting error estimates as predictors for steps which are far too long and by trusting the same error estimates in the presence of rapid changes of order. We believe that our Phases 1 and 2 are not only much cheaper, but also more reliable. The subsequent adjustment in Phase 3 is far more plausible than that used in the difficult circumstances of Sedgwick's code.

3.4. The algorithm-summary

In summary our algorithm for computing an initial step size when the user does not wish to intervene is as follows:

(i) *Phase 1.* We estimate the initial step from (3.4) where the weight in the norm is given by (3.5).

(ii) *Phase 2.* Starting with the step produced from Phase 1, use the Runge–Kutta method to take a step checking at each stage for which (3.12) is satisfied that (3.11) is also satisfied. (In these checks the norm $\|\cdot\|_a$ is defined by (3.8).) If (3.11) fails to be satisfied at any stage the step is terminated and the step size is reduced using (3.13). If (3.11) is satisfied at each stage then the local error is estimated in the usual way for the code and its magnitude (in the norm (3.2)) is checked against the local error tolerance τ. If the step would be a success then Phase 2 terminates otherwise the step size is reduced by a factor r. After a step size reduction Phase 2 is re-entered with the new step.

(iii) *Phase 3* We always enter Phase 3 with a successful step. If $1 \leqslant \alpha \leqslant r$ where α is the proposed scaling of the step, we terminate Phase 3. Otherwise we increase the step size in

accordance with (3.14) and repeat the step. Our general strategy on the following iterations is then:

if $1 \leqslant \alpha \leqslant r$,	terminate Phase 3;
if $\alpha > r$,	use (3.14) and repeat the step; and
if $\alpha < 1$,	use (3.15) and repeat the step.

3.5. User supplied initial step size

Many initial value codes for ordinary differential equations permit the user to specify an initial step size, and all the early codes demand it. Even with a reliable and efficient automatic starting procedure such as that described here, it is still advantageous to permit the user to supply an initial guess. This is so because there are circumstances where the user knows an on scale step size, and also there are degenerate situations where an automatic procedure is bound to be comparatively expensive. A convenient way to handle the user interface is that adopted by Gladwell [3]. The interval of integration is specified along with an input value of H. If $H \neq 0$, the code is to take H as the initial step. Otherwise, the code selects an initial step size automatically.

If the user does supply an estimated initial step size, it would be reasonable to treat it as a predicted step for normal integration. However, it is possible to rectify a bad estimate without much extra code as follows. We assume the user's step gives a valid error estimate, and hence if it fails, we try an optimal reduction constrained as in (3.15). If the user's choice leads to a successful step, we treat it just as if we had entered Phase 3. It is tempting on grounds of safety to enter Phase 2 with an estimated initial step size. On trying this we were reminded that the Phase 2 is intended to shorten H as necessary to get a credible error estimate and is often quite conservative. As a consequence, a good guess provided by the user may very well be shortened substantially and then be increased in Phase 3 to the general size of the input value. We prefer, then, to suppose that the user will not provide a value at all unless he can provide one for which the error estimate has some validity.

4. Numerical evaluation

An initial step size selection algorithm is notoriously difficult to evaluate, not least because in many codes either it does not form a readily identifiable segment of the code or it is trivial. Also, the aim of the algorithm is not always clear. It may simply produce a value to be tried, or may go to some lengths to produce a value likely to succeed. Some algorithms, like ours, seek a locally optimal step size. A few authors consider a local Lipschitz constant to be of independent interest and so a desirable byproduct of the algorithm.

The most recent and comprehensive investigation of the selection of the initial step size is that of Watts [14]. It is useful to consider how he measured the success of his approach. To demonstrate the efficiency of the step size h_s produced by his routine HSTART, Watts presents the total cost of integration of a set of problems for each of several initial step sizes, namely $h_s/100$, $h_s/10$, h_s, $10h_s$, $100h_s$. The fact that h_s results in the least total cost supports a claim that it is a good choice of initial step size. The difficulty with this method of assessing effectiveness is that if the code has a good step size adjustment algorithm, it will minimise the

effect of a poor initial step size. The results presented by Watts for a comparatively easy set of test problems with the effectiveness measured in this way do not show HSTART to be much better than what amounts to our Phase 1. In our view, the value of Watts' subroutine lies in its ability to handle difficult problems; his detailed discussion of individual problems is more useful for assessing HSTART than this tables showing efficiency.

An important difference between our study and that by Watts is that we want the locally optimal step size so insist on a step size which is at least on scale. Watts presents some evidence to the effect that his algorithm usually results in a reasonable approximation to the locally optimal value. He cannot go further because HSTART is provided with no detailed information about the method employed by the integrator. We must have this information, so it seemed natural to exploit the form of the method in our Phase 2. We agree with Watts that it is desirable to confine one's attention to a neighbourhood of $y(x)$, and this is realised effectively and efficiently in our Phase 2. It must be appreciated that because HSTART uses minimal information about the integration, it can be used as an auxiliary routine. The monitoring which we do in Phase 2 takes account of the specific formula and is built most naturally into the integrator of interest. The adjustment of the step size which we make in Phase 3 needs to be done in the integrator, too. Thus, we describe in this paper a methodology for initial step size selection rather than a subroutine.

We report here the qualitative phenomena which we have observed using our algorithm. We have performed a variety of tests designed to prove that the algorithm is robust, that it delivers an on scale initial step size, and that generally it is not more expensive to use than those algorithms used in earlier generations of codes. The test problems are those of the non-stiff and stiff test sets given in [5] and [2], respectively, and also some additional problems considered by Watts. We have run our code based on the DPS formulas [11] on all these problems with the initial step size algorithm described in Section 3.

Our experience for the non-stiff set with an absolute local error control and for a variety of error tolerances is generally that the step size computed by Phase 1 is not reduced in Phase 2 and also it results in a successful step when tried in Phase 3. In every case, the result of Phase 3 was on scale.

The stiff problems are more challenging, and perhaps extreme, for these procedures aimed at non-stiff problems. With the absolute error control specified in [2], we observed a behaviour similar to that found when solving the non-stiff problems. Such an error control may not be realistic for stiff problems in general, but these particular problems have been scaled so that it is not unreasonable for them. A possibly more realistic, and certainly more demanding, task is to treat these test problems with a relative error control. In this case the major difference observed is that in some cases Phase 1 returns a step size which is reduced sharply in Phase 2 to satisfy (3.11) and then a few iterations of Phase 3 produce an on scale initial step. We view this as acceptable behaviour when solving stiff problems as each phase is meeting its objectives. The fact that the algorithm performs more than adequately when faced with a stiff problem is an indication of its reliability and robustness.

In Tables 1–3 we present a summary of integrations of the non-stiff and stiff test sets where we have chosen $r = 10.0$ in the definitions of Section 3. On leaving Phase 2 we have found a successful step, H. It is predicted that the locally optimal step size is αH, $\alpha \geq 1$. If $1 \leq \alpha \leq r$, the starting step size procedure will terminate and the integration proceed from $a + H$. If $\alpha > r$ (> 1), the step size will be adjusted by the iteration in Phase 3 to a closer approximation to the

Table 1
Categorisation of non-stiff test set run with absolute error tolerance and $r = 10$

		Number of iterations of Phase 2		
		1	2	3
Step succeeded	$1 \leqslant \alpha \leqslant r$	72	4	0
	$\alpha > r$			
	0 fail in Phase 3	0	1	5
	1 fail in Phase 3	3	4	1

locally optimal value. Table 1 categorises the behaviour of the initial step size algorithm in the context of the DPS code. There are 90 test problems arising from integrating 30 non-stiff problems with local error tolerances $\tau = 10^{-1}, 10^{-4}, 10^{-7}$ in turn.

The six cases where there are three iterations at Phase 2 correspond to the two test problems which have zero initial conditions and zero initial slope, leading to the choice $b - a$ for the initial step size from Phase 1. The algorithm recovers quickly from this degenerate situation in Phase 2. Without that part of Phase 2 designed to take account of variations of f with respect to the independent variable, far more iterations would be required for these problems.

In Phase 3 for every problem there is at most one step size increase. There are two cases where the error estimate produced in Phase 2 is too large. These result in an immediate reduction to a successful on scale step size.

It is worth noting that there are a number of cases with one failed step after an initial successful step in Phase 3. These all result from choosing such a large scale factor α on entry to Phase 3 that the nature of the problem changes somewhat over the interval spanned by the new trial initial step and so the resulting local error is greater than predicted from the evidence available on entry to Phase 3.

Table 2 summarises the performance of the initial step size algorithm in the integration of the 30 stiff test problems. The entries correspond exactly to Table 1 and qualitatively the results are similar to those in Table 1. Again only one increase in step size ever occurs in Phase 3, and the one case of a multiple failed step can be attributed to 'optimal' step size reductions which are not fully justified.

In another test of the initial step size algorithm we achieve effectively a pure relative error control by setting $\text{THRES}_i = 10^{-10}$ in (3.2). Then, repeating the calculations summarised in

Table 2
Categorisation of stiff test set runs with absolute error tolerance and $r = 10$

		Number of iterations of Phase 2		
		1	2	3
Step succeeded	$1 \leqslant \alpha \leqslant r$	62	16	0
	$\alpha > r$			
	0 fail in Phase 3	0	0	0
	1 fail in Phase 3	0	6	5
	2 fails in Phase 3	0	1	0

Table 3
Categorisation of stiff test set runs with relative error tolerance and $r = 10$

			Number of iterations of Phase 2	
			1	2
	$1 \leqslant \alpha \leqslant r$		36	2
Step succeeded	$\alpha > r$	No. of times step size increased		
		1	3	0
		2	4	0
		3	7	0
		4	15	0
		5	10	0
		6	10	0
		7	2	0
	$\alpha > r$	No. of subsequent failures		
	once	1	0	1

Table 2 we obtain the results presented in Table 3. For the many problems where the initial conditions are zero, the code selects a very small initial step which in most cases is subsequently increased a number of times. The figures giving the number of times this step is increased are rather misleading. In total 51 test problems are in this category and there are a total of 216 iterations of Phase 3. Of these, just two of the iterations involved taking the maximum permissible increase in step size, r^3. There are cases where six or seven increases are required in Phase 3 to increase the step produced by Phases 1 and 2 to an on scale step. All twelve such cases occur at the most stringent tolerance, $\tau = 10^{-7}$. These cases are all qualitatively similar, and in the worst, a total increase in step size at Phase 3 of a factor of approximately $2.7 * 10^{10}$ is achieved in seven iterations with a mean increase factor of approximately 31 per iteration. Despite this relatively slow increase the algorithm is very successful in terms of cost. If the integration had been started with the step size output from Phase 2 and using the normal maximal increase per step of $r = 10.0$, then on integrating the worst case above the code would have progressed only about one-hundredth of the on scale step size taken by Phase 3 for the same cost in function evaluations. Two or three *additional* successful steps would be required to 'catch up' with the integration started using our initial step size. Finally, we observe that despite the relatively small changes made on each iteration of Phase 3, the step computed is 'on scale' for all our test problems. That is, the prediction for the second step of the integration is approximately equal to the computed initial step.

Clearly, the results presented in Tables 1–3 are dependent on our choice $r = 10$. This is somewhat larger than is often used in practice and so we have also run our code with $r = 5$ and $r = 2$ to test the robustness of the starting procedure. We obtain results which are qualitatively similar but with a greater tendency to require more iterations both of Phase 2 and of Phase 3 as r is reduced. As an example, we present in Table 4 the results corresponding to those in Table 1 but for $r = 2$. Observer the greater tendency towards step failures in Phase 3 with the small value of r.

Though the results presented in Tables 1–4 are very reassuring, they do not provide information about the robustness of our algorithm when used in extreme circumstances. An

Table 4
Categorisation of non-stiff test set run with absolute error tolerance and $r = 2$

		Number of iterations of phase 2				
		1	2	3	4	5
$1 \leqslant \alpha \leqslant r$		7	1	2	1	0
$\alpha > r$	No. of times step size increased					
	1	55	1	0	0	2
	2	1	0	0	0	2
	3	0	0	0	0	1
$\alpha > r$	No. of times stepsize increased followed by one subsequent failure					
	1	12	3	0	0	0
	2	0	0	0	0	1
	3	1	0	0	0	1

exception is Phase 1 which is fully tested in the 270 problems considered above. It is clear from our results that Phases 2 and 3 are able to deal with very bad outputs from Phase 1.

It is of concern that Phase 2 should not accept a step size which is too large due to, say, inadvertent numerical cancellation in the calculation of (3.11). To test the robustness of this phase, we discarded Phase 1 and entered Phase 2 with a step size set to $b - a$. All the non-stiff and stiff test problems were solved in this way. In every case Phase 2 succeeded in reducing this value quickly to a size suitable for entry to Phase 3. The vast majority of reductions in Phase 2 both here and in the results quoted in Tables 1–3 occur after just one stage (involving one evaluation of f) of the step of Phase 2. The value of Phase 1 has been amply demonstrated elsewhere. These computations show that our new algorithm of Phase 2 is a very efficient and effective way to improve the prediction of Phase 1.

More serious than a failure of Phase 2 to meet its objectives is that Phase 3 fail to produce an on scale initial step size. In particular we are concerned that Phase 3 not accept a step erroneously; that is, we are concerned that the error estimate should not underestimate seriously the magnitude of the local error. To study this matter we adopted the following strategy. For the problems of the non-stiff and stiff test sets and for a variety of local error tolerances we first calculated the initial step size with our algorithm. The code was then called with a user-supplied guess for the initial step size consisting of a thousandth part of the result of our algorithm. This second computation adjusts the step size input to Phase 3 to a value believed to be on scale. In all cases the two computations resulted in values which agreed to within a factor of r; apparently our algorithm was never deceived. The computations demonstrate the robustness of the overall algorithm and in particular that Phase 3 is entered with a step size for which the error estimate has some validity.

Finally we consider in more detail the behaviour of our algorithm when starting the integration of two problems considered by Watts [14]:

$$\text{X1: } y'' = -y - \operatorname{sgn}(y) - 3\sin(2x), \quad y(0) = 0, \quad y'(0) = 3,$$

$$\text{X2: } y' = 1/(1 + y^{1/2}), \quad y(0) = 0.$$

To reveal potential difficulties we use a threshold $\text{THRES}_i = 10^{-10}$, for all i, giving effectively a relative error control. For problem X1 the only difficulty to be expected is a sign change in y; of

course the initial point is at a sign change, but we are integrating away from it. With any reasonable error tolerance no other sign change will be encountered in the first step. With the threshold values chosen and for each of $\tau = 10^{-1}$, 10^{-4} and 10^{-7}, a small step size is chosen by Phase 1 which passes unchanged through Phase 2. Then the step size is increased a number of times in Phase 3 until an on scale step size is accepted. Qualitatively the behaviour is very similar to that for the test problems of Table 3.

When integrating problem X2 a small step size is estimated by Phase 1 and is accepted by Phase 2. Then, depending on the local error tolerance τ, Phase 3 may increase the step, but by quite small factors, before terminating with a successful step. However, this step size is not on scale for the remaining part of the integration. On the second and subsequent steps significant increases are made in the step size. The step size used over most of the integration is far larger than the initial step size returned by Phase 3, especially in integrations at the more relaxed error tolerances. These observations do not imply that the initial step size algorithm is failing in its aim; indeed the same behaviour is observed with HSTART. The explanation is that the second derivative of $y(x)$ has a singularity at the initial point. One consequence is that the method does not exhibit its expected order. It is some relief that the local error is not so severely under-estimated as to lead to a prediction of a far larger step size increase than is wise. Another consequence of the singularity is that whatever step size is appropriate for the initial step can scarcely be on scale for the remaining integration.

Acknowledgements

L.F. Shampine wishes to acknowledge the many interesting and productive discussions of initial step size selection with his colleague H.A. Watts. I. Gladwell and L.F. Shampine are grateful for the support of NATO research grant 898/83, without which their collaboration would have been difficult. The authors wish to thank N.J. Higham and a referee for their careful reading of the manuscript and useful comments.

References

[1] A.R. Curtis, Solution of large stiff initial value problems—the state of the art, in: D. Jacobs, Ed., *Numerical Software — Needs and Availability* (Academic, London, 1978) 257–278.

[2] W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of ODE's, *BIT* **15** (1975) 10–48.

[3] I. Gladwell, The NAG library ordinary differential equations chapter and short term plans for its extension, *SIGNUM Newsletter* **20** (1985) 35–40. (See also the specification of subroutine D02PAF in the NAG Library Manual for a more detailed discussion.)

[4] I. Gladwell, Initial value routines in the NAG library, *ACM Trans. Math. Software* **5** (1979) 386–400.

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

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

[7] B. Lindberg, IMPEX 2—a procedure for solution of systems of stiff differential equations, Rept. TRITA-NA-7303, Dept. Info. Processing, Royal Inst. of Tech., Stockholm, 1973.

[8] A.E. Sedgwick, An effective variable order variable step Adams method, Rept. 53, Dept. Comp. Sci., Univ. of Toronto, 1973.

[9] L.F. Shampine, Limiting precision in differential equation solvers, *Math. Comp.* **28** (1974) 141–144.
[10] L.F. Shampine, Lipschitz constants and robust ODE codes, in: J.T. Oden, Ed., *Computational Methods in Nonlinear Mechanics* (North-Holland, Amsterdam, 1980) 427–449.
[11] L.F. Shampine, Some practical Runge–Kutta formulas, *Math. Comp.* **46** (1986) 135–150.
[12] L.F. Shampine and M.K. Gordon, Some numerical experiments with DIFSUB, *SIGNUM Newsletter* **7** (1972) 24–26.
[13] H.J. Stetter, Tolerance proportionality in ODE-codes, in: R. März, Ed., Seminarberichte M.32 Sektion Math., Humboldt Univ., Berlin, 1980, pp. 109–123.
[14] H.A. Watts, Starting step size for an ODE solver, *J. Comp. Appl. Math.* **9** (1983) 177–191.