Global Error Estimation for Ordinary Differential Equations

L. F. SHAMPINE and H. A. WATTS

Sandia Laboratories

The user of a code for solving the initial value problem for ordinary differential equations is normally interested in the global error, i.e. the difference between the solution of the problem posed and the numerical result returned by the code. This paper describes a way of estimating the global error reliably while still solving the problem with acceptable efficiency. Global extrapolation procedures are applied to parallel solutions obtained by a Runge-Kutta-Fehlberg method. These ideas are implemented in a Fortran program called GERK, which is ACM Algorithm 504.

The Algorithm: Algorithm 504, GERK: Global Error Estimation for Ordinary Differential Equations. *ACM Trans. Math. Software 2*, 2 (June 1976), 200–203.

Key Words and Phrases: ordinary differential equations, initial value problems, global error estimation, Runge-Kutta-Fehlberg method, Fortran code GERK
CR Categories: 3.20, 5.17

1. INTRODUCTION

The user of a code for solving the initial value problem for ordinary differential equations is normally interested in his global error, i.e. the difference between the solution of the problem posed and the numerical result returned by the code. Codes do not attempt to control this global error directly and very few even try to estimate it. We have found that it is possible to estimate (but not to control) the global error reliably while still solving the problem with acceptable efficiency; this paper describes one way to do this.

Our objective is to write a code which will estimate global error and be as effective and efficient as possible. In Section 2 we consider some limitations inherent in the task. There are several general principles by which one might estimate global errors and there are special techniques [2, 14] as well which appear promising. Because efficient formulas for the latter have not yet appeared, we have given our attention to the general principles only. In Section 3 we describe some of the factors involved in forming an efficient, reliable method by each principle. The best scheme we have been able to construct uses the Runge-Kutta formulas of Fehlberg and

A version of this paper was presented at Mathematical Software II, a conference held at Purdue University, West Lafayette, Indiana, May 29–31, 1974.
This work was supported by the U.S. Energy Research and Development Administration.
Authors' address: Division 2642, Sandia Laboratories, Albuquerque, NM 87115.

the principle of global extrapolation. In Section 4 we describe a code, GERK, implementing this scheme. In Section 5 we present results about efficiency and reliability of the code and examples of its use.

2. INHERENT LIMITATIONS

Granted that one is going to estimate the global error, can one go so far as to control it? A little reflection shows the answer is generally no. The situation is governed by the behavior of the integral curves of the equation. Let us first examine a relation between the local error and the global error. Suppose y_n denotes the computed solution at x_n and $u(x)$ represents a local solution of the differential equation having initial value y_n at x_n. Then

$$\begin{aligned}\text{global error (at } x_{n+1}) &= y_{n+1} - y(x_{n+1}) \\ &= \{y_{n+1} - u(x_{n+1})\} + \{u(x_{n+1}) - y(x_{n+1})\}.\end{aligned}$$

The first term in braces is the local error while the second depends on the stability of the differential equation. More to the point, the second term is approximately (for small step sizes)

$$\{1 + h_n J_n\} * \{\text{global error (at } x_n)\}$$

where $h_n = x_{n+1} - x_n$ and J_n is the Jacobian of the differential equation at (x_n, y_n). The eigenvalues of J_n determine the stability of the differential equation. Thus the second term measures the sensitivity of the problem itself and not the method being used.

Suppose we have been keeping the global error below some specified tolerance through the point x_n. If the problem becomes unstable and the integral curves fan out at x_n so that the global error becomes unacceptable in the next step, one may not be able to obtain an acceptable error by merely changing the step size. Instead, the whole integration may have to be started anew with a smaller tolerance. The opposite situation occurs when the integral curves pinch together at b, the place where one wants a specified accuracy. In order to be efficient it is then necessary to allow the global error to be larger during the integration than is desired at b. Because of these two extreme possibilities we conclude that control of global error, in general, cannot be based satisfactorily on the local device of step-size adjustment. For these reasons we report the estimated global error and select the step sizes to control the local error in familiar fashion rather than try to control the global error.

Having to use a local error control is actually beneficial in several respects. From the preceding expressions it is clear that the effect on the global error of keeping the local error small depends on the differential equation itself. But if the equation is not mathematically unstable, controlling the local error will approximately control the global error. We believe the reliability of the global error estimates is greatly enhanced by step-size adjustment based on local error control, especially when working with the larger step sizes resulting from crude error tolerances. Local error control detects the onset of numerical instability and adjusts the step size so as to keep the computation stable. This has an important practical consequence. We consider, for example, two simultaneous integrations with the same step sizes

but with different methods. Because control of the local error maintains the step size at about the boundary of the absolute stability region when this is a problem, we must determine the step size by the integration with the less stable method. The contrary simply cannot be used if stability is limiting the step size.

3. SELECTING A METHOD

All the general principles for global error estimation that we consider involve two parallel, independent integrations. One possibility is to use the same step sizes but different methods. Another is to use the same methods but different step sizes. A possibility closely related to the latter is to integrate the problem twice with the same code and different tolerances. For codes implementing a single method this can be regarded as a variant of the second scheme, but in codes like the variable order Adams codes, both different methods and different step sizes are used in the two integrations.

The first way of estimating the global error we examine in detail is the method of repeated integration or reintegration. One solves the problem twice with the same code but with the error tolerance reduced, for example, by a factor of 1/10. What are some of the advantages of this technique? First of all, it is a familiar technique which can be applied with any code for the initial value problem; so no additional software is necessary for the computing library. Second, one can always return to a computation at a later date and check its accuracy if it seems necessary. And, third, reintegration may work quite well in the presence of discontinuities—an advantage not shared by other methods we consider. What are some of the disadvantages associated with repeated integrations? No estimate of the error is available until the problem has been integrated twice. One can only reliably estimate the accuracy of the less accurate result—this is probably the most serious drawback of the technique. Reintegration may not be feasible—for example, if the integration process is imbedded in some larger and quite complicated program. This procedure for estimating global accuracy is relatively expensive. Let us suppose the tolerance is reduced by a factor of 1/10 and let us compare the total cost of solving the problem twice as opposed to just once at the larger tolerance. For a good variable order code the cost only slightly more than doubles because the additional accuracy is achieved by variation of the order rather than reduction of the step size. For fixed order codes the cost rises more rapidly, e.g. for order four the cost increases by a factor of about 2.6. The theoretical support of this technique is not as good as for other techniques we study, and the estimates are not as good. Unless a code has been written with the technique in mind, global error estimation by reintegration can be very risky. The algorithm may be unduly conservative or so poorly implemented that reductions in the tolerance fail to produce proportional changes in the error or, worse yet, fail to cause any change at all. We have seen many codes which are not reliable in this way. A case in point is the rational extrapolation code DESUB. In the tests of [13] it was not uncommon for this code to produce identical results with tolerance reductions of up to three orders of magnitude.

The next procedure of estimating global error that we discuss is that of integrating using two different order methods but using the same step size for both. Although this process can be put on a sound theoretical foundation, a cursory ex-

amination shows that unless there is some special relation between the methods used, its performance will be unsatisfactory on a nonasymptotic basis. Also, this technique shares a disadvantage of reintegration in that the accuracy of the less accurate result only can be estimated. Furthermore, because of stability considerations it is important to choose the right one of the two parallel solution formulas to govern the selection of the step size; the formula used for this purpose must have the smaller stability region. We have experimented with this procedure using parallel solutions having accuracy $O(h^p)$, $O(h^{p+1})$ where the more accurate solution is obtained by performing local extrapolation [9]. We give an example of this method later.

The last way of estimating global errors we study involves parallel integration, one being carried out with half the step size of the other, using the same basic method. It currently appears that only the method of repeated integration is feasible for the powerful variable order Adams codes [11]. Accordingly, we shall concentrate on a fixed order, one-step code. By using global extrapolation on the parallel solutions we can estimate the global error of the *more accurate result* as is justified in the classic text of Henrici [4]. Let $Y(x,h)$ denote an approximate solution obtained at the point x using a variable step size with $h = \theta(x)H$ where H is constant and $0 < \theta(x) \leq 1$, let $Y(x,h/2)$ be the approximate solution generated at x using the step size $\theta(x)H/2$, and let $y(x)$ be the true solution. Then the global error for a numerical method of order p is given by

$$E(x) = Y(x,h/2) - y(x) = (H/2)^p e(x) + O(H^{p+1}),$$

where $e(x)$ satisfies a certain differential equation. Implicit in this representation is the assumption that the solution has continuous derivatives of order at least $p+2$ in the region of interest. In this case, global extrapolation becomes valid and we get

$$E(x) \doteq (Y(x,h) - Y(x,h/2))/(2^p - 1).$$

There has been other discussion of global extrapolation [7] but no one seems to have studied those factors relevant to writing a robust, efficient code.

We have tried to construct the most efficient procedures we could using these principles. Many possibilities can be discarded on general grounds, but in the case of close competitors we have written codes to compare them experimentally. Our colleague M. K. Gordon made available to us other experimental work which assisted us in our choice. It appears that global extrapolation using one of the Runge-Kutta-Fehlberg (4, 5) possibilities [3] is the most satisfactory in terms of accuracy, reliability, and efficiency. In [12] we have compared a number of Runge-Kutta processes with respect to their local error estimators and subsequently have tested others in the same way. We have done very extensive tests of the better Runge-Kutta processes as to their relative efficiency. The Fehlberg scheme we have chosen has been as good or better than all others we have considered. In what follows we compare its use according to the various principles for global error estimation. Naturally we have also considered methods peculiarly suited to a given principle, and in a few cases we shall comment about this, but in general no other scheme compares to this one and we need not report on the discards.

Before discussing how the Fehlberg scheme is used for global error estimation we shall briefly describe it. At a cost of 6 function evaluations each basic step, distinct fourth and fifth order accurate solution values Y_4 and Y_5 are computed. Letting e_4, e_5 represent the corresponding local truncation errors, we have $Y_4 = Y + e_4$ and $Y_5 = Y + e_5$, where Y is a local solution of the differential equation and $e_4 = O(h^5)$, $e_5 = O(h^6)$. Fehlberg has chosen the parameters in the Runge-Kutta formulas so that the nine coefficients making up the $O(h^5)$ term in e_4 are chosen to be about as small as possible consistent with avoiding contradictions in the equations of condition. An estimate ϵ of the local error in Y_4 is then computed as

$$\epsilon = Y_4 - Y_5 = e_4 - e_5 = O(h^5).$$

From our studies of this Fehlberg method the following important results emerged. In comparing the local error estimator with those of [12], we have found it to be as accurate, asymptotically, as the Ceschino-Kuntzmann estimator discussed there (the most accurate of those considered). On a nonasymptotic basis, we saw the tendency of the estimator ϵ to underestimate e_4, the local error in Y_4, about 65 percent of the time. However, even in the nonasymptotic step-size regime, the Y_5 solution consistently proved to be the more accurate (except for grossly large step sizes) and the estimate ϵ typically overestimated e_5, the local error in Y_5. Finally, the stability region for the Y_5 formula is about 19 percent larger (uniformly) than the stability region for the Y_4 formula. For these reasons, we actually use the fifth order formulas, termed local extrapolation [9]. In view of our primary intent of assessing the global accuracy of the locally extrapolated solution which is reported to the user, one cannot make the usual objection about the extrapolated result being of unknown accuracy (though asymptotically more accurate than required).

The global extrapolation algorithm adopted for the GERK code advances one solution over a basic step using local error control and, once this is acceptable, the other (parallel) solution is advanced over two half-steps using the same formula. Since each application of the Fehlberg formula costs 6 functions evaluations, the cost per basic step of this global extrapolation process is 18 derivative function evaluations.

In [12] a scheme due to England showed up very well. Because it appears particularly suited to global extrapolation we shall indicate why it was not selected. This procedure uses a fourth order Runge-Kutta method requiring 4 function evaluations to advance the solution two basic steps while estimating the local error accumulated over the pair of steps. The resulting cost is then 9 function evaluations. On the surface the global extrapolation technique for estimating global error seems tailored for application to the England process, or to any method which uses the doubling procedure for estimating local error. This is because it would be easy to apply the same basic Runge-Kutta process in computing a parallel solution over the step length which is double that used for the other solution. The corresponding global extrapolation scheme using the England formulas would cost 13 derivative function evaluations. However, this scheme can result in precisely the stability difficulty mentioned earlier. In fact, when numerical stability is a problem the global error estimate computed with this scheme is of the order of magnitude of the solution obtained with the larger step size—ridiculously large because the numerical instability causes the solution to explode. On the other hand, the true

global error of the solution to be reported is perfectly acceptable, usually being smaller than that requested. Furthermore, on problems where numerical stability is not a factor we have observed that this England process is from 30 to 50 percent less efficient than the Fehlberg scheme described above. This is partly because the coefficients of the error term in Fehlberg's scheme are quite small and partly because we cannot extrapolate the England scheme when it is used in this way (hence it is of lower order).

Next let us consider a procedure for estimating the global error which uses the same step size but different order methods. Suppose we compute parallel integrations using the fourth and fifth order Fehlberg formulas in an independent fashion for obtaining global solutions S_4 and S_5, say. It is necessary to select the S_4 solution process for the local error testing and adjustment of the step-size because, in addition to not having local error estimates of the fifth order approximation available, the stability region for the fourth order formula is also the smaller. This global error estimation procedure costs 12 function evaluations per basic step. As pointed out earlier, one gets an estimate of the error in only the less accurate solution S_4 and so this must be the reported solution. When stability is limiting the step-size choice, this procedure is more efficient than the adopted global extrapolation scheme due to using 6 fewer function evaluations per step and the fact that additional accuracy can be achieved for such problems with little extra cost. When stability does not limit the step-size choice, we have observed that this procedure is from 40 to 70 percent less efficient than global extrapolation. This sharp reversal in the relative efficiency of the two procedures which use the same integration formulas is surprising; so let us explain this behavior. If step size and error control were performed on the "S_4 solution" for both procedures, we would see identical step-size sequences for the two codes applied to the same problem with the same local tolerance parameters. In this circumstance, the S_5 solution would be the same as the $Y(h)$ solution in the global extrapolation process but less accurate than the reported solution $Y(h/2)$. However, if the step-size sequence for the scheme of this paragraph is altered to use $h/2$, then S_5 would be identical to $Y(h/2)$. The comparative cost in achieving this result is 24 versus 18 function evaluations, which accounts for a 33-percent cost increase over global extrapolation. The other factor present is that step size and error control are applied to the $Y(h)$ solution in the global extrapolation process; that is, local error estimates are based on advancing the global S_5 solution over a single step. Since this is a more accurate solution than S_4, at least asymptotically, it typically leads to a more appropriate step-size sequence. This could explain the additional factor which makes the adopted global extrapolation scheme even more efficient than at first apparent when compared to the scheme using the different order methods just described. It turns out that both schemes are about equally reliable in estimating the global error.

Last, let us compare the use of repeated integration for estimating the global error. Suppose we construct a code which computes the locally extrapolated solution S_5 from the Fehlberg formulas and returns this solution to the user. This is identical to the solution $Y(h)$ being carried along in the global extrapolation process. However, the global extrapolation process returns the more accurate solution $Y(h/2)$ along with an estimate of its global error. As in the previous example, the case when stability limits the choice of step size is rather special. It is an important

point that in these circumstances all tolerances can be achieved with only slight changes in the step size. Thus the second integration (for the reintegration process) is achieved with virtually the same cost as the first which can, again with little extra cost, produce a solution comparable in accuracy to $Y(h/2)$. Hence, when stability is a factor, reintegration becomes more efficient than global extrapolation and the cost ratio of the respective function counts can approach 12/18. When stability is not a factor, and comparable accuracies are achieved in the reported solutions along with corresponding global error estimates, reintegration is more expensive than global extrapolation by a factor of roughly 1.5 to 2, about 1.7 from an asymptotic analysis. To justify these claims, let us suppose that the tolerance is reduced by a factor of 1/10 for the reintegration procedure and let us examine what corresponding step-size change occurs. While the propagated solution, S_5, has a local error which is proportional to h^6, the step-size adjustment is based on the local error estimate of the unextrapolated solution value which is proportional to h^5. In reducing the error by 1/10, the step sizes would be expected to decrease by a factor of approximately $10^{1/5} \doteq 1.6$. Also, since reintegration provides an estimate of the error in only the less accurate solution, we must perform the basic integration on the step-size sequence corresponding to $h/2$ for the global extrapolation process. These comments show that the cost comparison becomes 18 versus 12*(1+1.6) in favor of the global extrapolation process.

4. A CODE

We now discuss the general design of our code which follows the principles described in [10] but with some important improvements. We use the error per step criterion and local extrapolation which is very efficient. In addition, the code estimates an initial step size by hypothesizing that the error in a fourth order start will be h^4 times the error in a zero order start. Thus, we basically choose h from the relation

$$| h^5 | \, ||y'_{\text{initial}}/\text{tol}||_\infty = 1$$

where tol is a requested tolerance vector and the division is taken component-wise whenever it is properly defined. Otherwise, we choose the interval length when $y' = 0$ and 26 units of roundoff in the larger of x_{initial} and the interval length when tol = 0.

After both successful and unsuccessful steps a new step size is chosen by a locally optimal step-size strategy,

$$h_{\text{new}} = 0.9(|| \epsilon/\text{tol} \, ||_\infty)^{-1/5} h_{\text{old}},$$

where ϵ is an estimate of the local error. The constant 9/10 was obtained by considerable experimentation and represents a conservative factor for preventing unnecessary step failures. This corresponds to aiming at an error of about 0.59 times tol while accepting a step with a local error estimate of the size of tol. Practical limits on the change in the step size are enforced to smooth the step-size selection process and to avoid excessive "chattering" on problems having discontinuities. We do not permit the step size to decrease by more than a factor of 1/10 or to increase by more than a factor of 5. Furthermore, after a step failure, the step size is not allowed to increase on the next attempted step. While this makes the code more efficient on problems with discontinuities, it also makes the code more effec-

tive in general since local extrapolation is always being used and the error estimate may be unreliable or unacceptable when a step fails. The code presented here also looks ahead two steps to output points to avoid drastic changes in the step size and thus lessens the impact of output points on the code.

The asymptotic analysis in the global extrapolation formula ignores the effects of roundoff. Clearly, as one approaches a limiting precision determined by the machine and code being used, the global error estimate becomes unreliable. Because of this it is very important to detect limiting precision with its potential unreliability. The matter has seen some attention in the literature [8] and we include similar tests which serve as a partial remedy. To begin with, the step size is not permitted to become smaller than 26 units of roundoff in x, so that the various arguments in the Fehlberg formulas can be distinguished for the integration using the smaller step size. Actually, the limiting precision difficulties arising from impossible accuracy requests are treated a little more simply here than in [8]. The code always requires (and adjusts if necessary) the relative error tolerance to be at least as large as the machine dependent level $32u + 3*10^{-11}$. Here u is the computer unit roundoff which is defined as the smallest positive value such that $1+u$ is greater than 1.

A low order method, such as the one used here, is generally incapable of achieving accuracies near the unit roundoff level on computers with long word lengths such as the CDC 6600. Furthermore, it becomes very inefficient. In our experience (*cf.* the tests of [13]) we have found it reasonable to cut off requested accuracies at a relative error tolerance of 3×10^{-11} for most problems when working with word lengths of 14 decimal digits or more. If the word length is quite short, it will limit the accuracy possible; so we need another cutoff level. Our experiments lead us to believe that 32 units of roundoff constitutes an ample cushion for most problems when using a short computer word length. While we can expect some contamination due to limiting precision difficulties in these circumstances, the effects should not invalidate the global error estimate except in occasional instances. Combined, these two cutoff levels lead to the choice of $32u + 3*10^{-11}$ for limiting the relative error tolerance for the code. We wish to emphasize that there may be other limitations imposed by the precision and that the above controls are simple necessary conditions. For other devices for controlling the effects of limited precision and propagated roundoff the reader is referred to [1, 15].

The code, GERK, we present is heavily commented with regard to its use, so we shall not reproduce the discussion here. However, a couple of remarks are in order. Although the code will typically be used to integrate from a to b, it can be used as a one-step integrator to advance the solution a single step in the direction of b. Upon each return an estimate of the global error in the solution at the current value of x is provided. If the stability properties of the differential equation are such that local errors do not significantly accumulate, the user can usually expect the global errors to be somewhat smaller than the input local error tolerances. This is because of reporting the more accurate solution being computed with one-half the basic step size. We had originally considered increasing the tolerances input by a factor of 32 but this proved to be somewhat dangerous on occasion and so we decided against it. Also, the user should recognize that the global error is zero upon initialization and so after the integration reaches an output point he should

not restart the code (IFLAG = ± 1) unless he truly wants to reinitialize the global error to zero at that point. Finally, it is worth emphasizing that the global error appraisal is an absolute comparison which estimates the difference $y_{\text{computed}} - y_{\text{true}}$.

A couple of situations deserve further comment. On mildly stiff problems (when stability limits the step-size choice), the global error estimate will often be smaller than the requested local error while the true global error will be even smaller than the estimate. We attribute this behavior to the fact that the solution on the half-step is not being troubled by the stability difficulty as much as on the full step. As a result the solution for two half-steps achieves more accuracy than is normally expected. When this occurs, the global error estimate is likely to provide an overestimate of the true global error. Recall also that on mildly stiff problems the global extrapolation scheme of estimating global errors was considerably less efficient than several alternatives.

If the code is tried on a problem in which discontinuities are present in the derivative function, the user should not expect to get good global error estimates once the discontinuity is encountered because the theory is simply no longer valid. Moreover, because the step size and error control are based on the approximation over the larger step, the reported solution could straddle a slightly shifted point of discontinuity resulting in a less accurate value than expected. This phenomenon cannot occur if the point of discontinuity is determined solely by the independent variable, but the possibility does exist if the dependent variables define the discontinuity. This situation represents a departure from the expected behavior of the associated Runge-Kutta code not estimating global errors.

5. PERFORMANCE OF THE CODE

Thorough testing of the GERK code presented has shown it to be reliable and surprisingly efficient. Of course, a very efficient and effective basic Runge-Kutta algorithm is the foundation of this code. In fact, GERK was constructed from a code called RKF45 which in the comparisons of [13] was found to be the most effective code when the differential equations are very cheap to evaluate and if low to medium accuracy is desired. Both codes, RKF45 and GERK, are members of a systematized collection of codes, DEPAC, currently being developed at Sandia Laboratories for solving ordinary differential equations.

It is interesting to examine the efficiency of GERK relative to RKF45. To begin with, GERK requires about three times the number of derivative evaluations that RKF45 does when using the same integration tolerances—roughly 18 versus 6. However, the solution returned by GERK is the more accurate one. If the step-size sequence used by RKF45 were altered by the factor of 1/2 as done in GERK, the solution obtained would be identical to the one reported from GERK, resulting in an efficiency factor of 18/12 when the codes achieve the same accuracy. Because this implies a possibly unnecessary restriction on RKF45, we would expect a somewhat higher efficiency factor in practical usage. To pursue this matter further, and to perform additional tests on GERK, we have subjected the code to the same tests as in [13], efficiency of function evaluations being primarily measured from solving the set of 25 test problems in Hull et al. [5] over the range of tolerances 10^{-2},

$10^{-3}, \ldots, 10^{-12}$. For each problem and each code the 11 data points ($\log_{10}$ global error achieved versus number of derivative evaluations) were fitted with various degree polynomials in a least squares sense. In each case the data fit was interpolated to obtain the number of derivative evaluations to *achieve* errors of 10^{-2}, $10^{-3}, \ldots, 10^{-12}$. Now for each error we computed the function count ratios of the two codes and, finally, these values were averaged over the ensemble of problems.

From these statistical analyses we obtained a remarkably uniform pattern—about 60 percent more derivative evaluations are required of GERK than of RKF45 to achieve the same accuracy over the entire accuracy spectrum of 10^{-2}, $10^{-3}, \ldots, 10^{-12}$. But this is not the complete story, as overhead cost may be quite important. In the study [13] we computed overhead cost *per derivative evaluation* as a linear fit in the number of differential equations. The comparison between GERK and RKF45 showed a decline in overhead cost (per derivative evaluation) for GERK of roughly 30 percent. Hence, putting these two factors together, examination of the total cost ratios of the two codes reveals that the increase in cost due to using GERK ranges from only 20 percent to 60 percent more than that of RKF45.

To gain some feeling about the role of overhead cost and when it is important, we have shown in Table I total cost ratios of GERK versus RKF45 to achieve the same accuracy for several numbers of differential equations and costs per equation. We have taken the unit of cost to be 25 μsec, which is roughly the expense of computing a trigonometric function, exponential, square root, or about 10 floating point additions sequentially on the CDC 6600. While these results are admittedly machine dependent, we believe them to generally reflect the relative efficiencies of the two codes.

We have collected many statistics on the reliability of the global error estimating capability of GERK when applied to the test problems of [5] and to other test problems. There are some troublesome matters which must be sorted out in order for the summaries we shall present to be meaningful. The effects of these special difficulties arise from limiting precision (step sizes too small), asymptotic estimates not being valid (step sizes too large), mildly stiff differential equations, global errors approaching zero, and some components being insignificant compared to the largest. We have already discussed the first three items; the latter two merely cause nuisances in compiling useful summaries of the results observed.

Table I. Total Cost Ratios of GERK Versus RKF45

Cost per equation (units of 25 μsec)	Number of equations: 1	5	10	∞
0.1	1.2	1.3	1.3	1.3
0.5	1.2	1.3	1.4	1.4
1.0	1.3	1.4	1.4	1.5
2.0	1.4	1.5	1.5	1.5
10 0	1.5	1.6	1.6	1.6

We compared the global error estimates to the true global errors at each of the designated output points used in the study [13]. For each component let us define

$$r = \text{global error estimate/global error},$$

when the numerator and denominator are nonzero. Because of the items mentioned above we occasionally encounter unimportant very large (and small) values of r as well as negative values, for which we have accumulated separate statistics. We have observed that about 25 percent of the total comparison (including all components being checked) yielded negative values of r. About 80 percent of these were concentrated at the ends of the tolerance range—the first three and the last three tolerances. Also, about 70 percent of the negative values of r were obtained on the problem set Class C of [5]. Zero values of the global error estimate and global error showed up on about 2 percent of the total number of comparisons with virtually all cases occurring on the Class C problems. This set happens to include problems exhibiting several of the difficulties already pointed out, namely mild stiffness, a change in sign of the global error, and insignificantly small components. For these reasons we have excluded the statistics from this set of problems in the results of Table II.

In order to obtain more meaningful averages it was necessary to compute the exponents $|\log r|$ ($|\log |r||$ when $r < 0$). Now at each of the designated output points we computed the largest deviations (examining all components) of the exponents from zero. For each tolerance these largest discrepancies were then averaged over the entire set of output points and finally over the ensemble of problems being considered. The averaged values of the exponents are used to compute the factors of $|r|$ shown in Table II. We have separated the results of positive and negative r.

To gain some additional confidence in the performance of the code over all the problems (including the Class C set), we monitored the ratios of maximum global

Table II. Comparison Factors from Global Error Estimates Versus Global Errors

−log (tolerance)	Positive ratios	Negative ratios
2	3.7	83.
3	3.6	13.
4	2.8	5.7
5	2.1	6.4
6	1.8	8.3
7	1.7	4.1
8	1.5	5.9
9	1.3	3.5
10	1.8	6.2
11	3.7	4.6
12	4.1	5.1

Table III. Ratios of Maximum Global Error Estimates Versus Maximum Global Errors

−log (tolerance)	Average	Maximum		Minimum	
2	.3	10.	(1)	.03	(2)
3	.5	3.2		.03	(1)
4	.6	3.2		.08	(1)
5	1.0	2.5		.5	
6	1.1	3.3		.7	
7	1.1	20.	(1)	.8	
8	1.1	2.9		.8	
9	1.0	1.4		.6	
10	1.2	4.5		.5	
11	.4	100.	(2)	.1	
12	.2	50.	(1)	.04	(6)

error estimates to maximum global errors. For each tolerance the maximum and minimum ratios and averaged values were computed over the ensemble of problems. These results are given in Table III. The numbers in parentheses indicate the number of times a ratio differed from 1 by at least a factor of 10.

We shall include two examples of the code's behavior. The first example is a mathematically unstable problem,

$$y'(x) = 10(y-x^2), \quad y(0) = 0.02,$$

which has the general solution

$$y(x) = 0.02 + 0.2x + x^2 + ce^{10x}.$$

We solve this problem on the interval [0, 2] using pure relative error control and monitor the global error at the end of each step using the code's one-step integrator mode. The following notation is used in Table IV:

d = factor which shows the largest discrepancy between the true and estimated global errors, global error estimate = d* true global error,
ND = number of derivative evaluations.

Tables IV and V

−log (tolerance)	True global error	d	ND
	Table IV. Mathematically Unstable Problem		
1	−7.2 (+4)	.11	91
2	−7.4 (+3)	.38	150
3	−4.2 (+2)	.68	314
4	−3.1 (+1)	.83	517
5	−2.9	.90	771
6	−2 9 (−1)	.94	1021
7	−3.0 (−2)	.96	1348
8	−3 1 (−3)	.97	2050
9	−3.1 (−4)	.98	3228
10	−3.3 (−5)	.86	5136
11	−1.2 (−5)	1.74	6522
12	−1.2 (−5)	1.74	6522
	Table V. Restricted 3-Body Problem		
1	−2.6 (+1)	−.03	193
2	−4.6	−.03	810
3	8.7 (−2)	−.03	1055
4	1.3 (−4)	.30	1506
5	1.3 (−5)	.64	2191
6	1.0 (−6)	83	3269
7	5 9 (−8)	.89	4873
8	−1.1 (−8)	1.01	7041
9	−8.8 (−10)	.77	11060
10	−2.5 (−10)	−.26	16373
11	−2.5 (−10)	−.52	20274
12	−2 6 (−10)	− 53	21665

In Table IV the number in the parentheses indicates the exponent of 10 which is associated with the corresponding value in the table. The global error shown is the maximum and occurs at 2.

Note how well the estimates monitor the unstable growth, as indicated by d being near 1. Furthermore, the global error decreases rather uniformly by a factor of about 1/10, indicating that repeated integration for estimating the global error would be rather successful with the underlying Runge-Kutta scheme. In fact, when the global error estimates for GERK are in doubt, one could apply GERK in the reintegration process for a highly reliable estimation of the global errors. Last, we see that limiting precision difficulties have been encountered with the smallest tolerances but that the code is performing satisfactorily.

Our next problem is the restricted 3-body problem [6],

$$y_1'' = 2y_2' + y_1 - \mu^*(y_1+\mu)/r_1^3 - \mu(y_1-\mu^*)/r_2^3,$$

$$y_2'' = -2y_1' + y_2 - \mu^* y_2/r_1^3 - \mu y_2/r_2^3,$$

$$r_1 = [(y_1+\mu)^2+y_2^2]^{1/2},\ r_2 = [(y_1-\mu^*)^2+y_2^2]^{1/2},\ \mu = 1/82.45,\ \mu^* = 1 - \mu,$$

$$y_1(0) = 1.2,\ y_1'(0) = 0,\ y_2(0) = 0,\ y_2'(0) = -1.04935750983032.$$

We solve this problem over the first period, $P = 6.19216933131964$, and use the interval oriented mode to estimate the global error only at $x = P$ since we do not have the true solution interior to this interval. This is a severe test of a code's step-size control (up to three orders of magnitude in the variation between the minimum and maximum step size were noted in the course of this integration). Also, for crude error tolerances it is rather easy to get off onto other integral paths. This problem shows rather markedly the three ranges of global error estimate reliability which may be encountered on any given problem. First, if the tolerances are too crude and the working step sizes are too large, one easily loses the correct solution curves. Under these circumstances the asymptotic analysis for global extrapolation is simply not valid and our error estimates are poor. In the second range the theory may be successfully applied and yields excellent agreement between the true and estimated global errors. In the third range the tolerances are at or near the machine and code limiting precision capabilities. In this case roundoff contaminates the solution to the level where the global error estimates are again incorrect. In Table V we follow the notation of Table IV except that absolute error control is used for this problem and we show the maximum global error of all the components and the factor d of the corresponding global error estimate.

Recall from the definition of d that the minus sign indicates that the estimate had the wrong sign of the true global error. Starting with the tolerance of 10^{-5} and continuing through 10^{-9} we obtain quite good results. Thus we see rather clearly the three ranges of reliability of the global error estimator. However, it should be appreciated that this is quite a difficult problem. Also the reader should keep in mind that the errors are measured only at the end of the period; so the errors reported could be smaller than the worst errors committed during the integration. Last, we have defined the three ranges of reliability possible with the global error estimates in rather loose terms, using the words "tolerances too crude," etc. The results are problem dependent but their reliability is generally quite satisfactory.

Table VI. Ratios d of the Estimated to the True Global Error for Problem 1, the Mathematically Unstable Problem of Table IV, and Problem 2, the 3-Body Problem of Table V, Obtained Using PDP-10 and IBM 370 Computers in Single Precision

	Problem 1		Problem 2	
−log (tolerance)	PDP-10	IBM 370	PDP-10	IBM 370
1	.11	.11	−.03	−.03
2	.38	.38	−.03	−.04
3	.69	.65	−.03	−.01
4	.83	.39	.30	.20
5	.88		.54	
6	3.6		.43	
7	−3.2		.13	

The word length of the machine used may limit the useful range of tolerances. In Table VI we report the results for the two example problems when computed on a PDP-10 which has about 8 decimal digits and an IBM 370 which has about 7. The difference in performance is greater than the word lengths suggest because the former is a binary machine with rounding and the latter a hexadecimal machine with chopping. It is clear that on the difficult 3-body problem one should use double precision on the IBM System 360 and 370 machines and others with similar arithmetic characteristics. On the other hand, we have already noted the code should not be used in double precision on the CDC machines and others with similar characteristics. Because single precision codes are the more transportable we have chosen to provide a single precision version of GERK.

ACKNOWLEDGMENT

The authors would like to thank Prof. Ivo Babuska of the University of Maryland for getting them interested in this study and Dr. Melvin Scott for his continued interest in the work. We also wish to thank Mr. Imre Farkas of the University of Toronto for providing the computations made on an IBM 370.

REFERENCES

1. Blum, E.K. A modification of the Runge-Kutta fourth order method. *Math. Computation 16* (1962), 176–187.
2. Butcher, J.C. The effective order of Runge-Kutta methods. Conf. on the Numerical Solutions of Differential Equations, Lecture Notes in Mathematics No. 109, Springer Verlag, 1969, pp. 133–139.
3. Fehlberg, E. Low-order classical Runge-Kutta formulas with step-size control and their application to some heat transfer problems. NASA Tech. Rep. TR R-315, George C. Marshall Space Flight Center, Marshall, Ala.
4. Henrici, P. *Discrete Variable Methods for Ordinary Differential Equations.* Wiley, New York, 1962.
5. Hull, T.E., Enright, W.H., Fellen, B.M., and Sedgwick, A.E. Comparing numerical methods for ordinary differential equations. *SIAM J. Numer. Anal. 9* (1972), 603–637.
6. Krogh, F.T. On testing a subroutine for the numerical integration of ordinary differential equations. *J. ACM 20*, 4 (Oct. 1973), 545–562.

7. LETHER, F.G. The use of Richardson extrapolation in one-step methods with variable step-size. *Math. Computation 20* (1966), 379–385.
8. SHAMPINE, L.F. Limiting precision in differential equation solvers. *Math. Computation 28* (1974), 141–144.
9. SHAMPINE, L.F. Local extrapolation in the solution of ordinary differential equations. *Math. Computation 27* (1973), 91–97.
10. SHAMPINE, L.F., AND ALLEN, R.C. *Numerical Computing: An Introduction.* Saunders, Philadelphia, Pa., 1973.
11. SHAMPINE, L.F., AND GORDON, M.K. *Computer Solution of Ordinary Differential Equations: The Initial Value Problem.* Freeman, San Francisco, Calif., 1975.
12. SHAMPINE, L.F., AND WATTS, H.A. Comparing error estimators for Runge-Kutta methods. *Math. Computation 25* (1971), 445–455.
13. SHAMPINE, L.F., WATTS, H.A., AND DAVENPORT, S.M. Solving non-stiff ordinary differential equations—the state of the art. To appear in *SIAM Rev.*
14. STETTER, H.J. Local estimation of the global discretization error. *SIAM J. Numer. Anal. 8* (1971), 512–523.
15. VITASEK, E. The numerical stability in solution of differential equations. Conf. on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics No. 109, Springer Verlag, 1969, pp. 87–111.

Received August 1975