Annals of Numerical Mathematics 1 (1994) 423-437 

# Issues in the design of a multistep code

Fred T. Krogh*

*Jet Propulsion Laboratory, California Institute of Technology,*
*4800 Oak Grove Drive, Pasadena, CA 91101, USA*

This paper summarizes things the author considers important from his past work, and presents some new ideas for use in the implementation of multistep methods. An effort has been made to include material of interest to those of a theoretical bent, as well as to those whose primary interest is in the implementation of methods.

**Keywords:** Numerical solution of ordinary differential equations.

## 1. Introduction

This paper summarizes my opinions on the integration of differential equations. I hope the following groups find something of interest in what follows.

For those interested in the implementation of multistep methods, a new approach to starting should be of particular interest. The direct integration of higher order equations, and a test for noise in the computation are covered briefly. Ideas not previously published on order selection and the treatment of discontinuities and constraints are outlined. A code planned for the integration of stiff and differential algebraic systems, delay equations, and of course the efficient integration of nonstiff differential equations is discussed briefly.

For those interested in one step methods, this paper offers the chance to spy on some of the competition. It is generally agreed that multistep methods work best if derivative evaluations are expensive. Research on one step methods has both improved their efficiency, and provided for more flexibility in output. Still, if one were to write a one step code with the generality described here for a multistep code, it is liable to suffer on integration overhead and code complexity to the point that it would have lost the big advantages one step methods possess.

For those interested in theory, the next section is intended for you. In addition,

* The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

a few comments appear in **bold face** to attract your attention. I am truly in awe of the theoretical results obtained concerning the numerical solution of ode's. But, have not had occasion to use your work, unless it was a result you obtained in the process of writing a code. When needing a result of a theoretical nature while developing a code, I have been on my own. Based on a sample of two other code developers, this is probably the typical situation. As one whose interests and talents are algorithmic rather than mathematical, I would like to see more help from those with the opposite inclinations.

Finally, there is an approach to developing algorithms that I have found useful not only for ordinary differential equations, but also for nonlinear least squares and quadrature. This approach involves solving very simple problems for which one knows the desired behavior based on theoretical considerations; watching closely what the algorithm does concerning such things as estimating errors and selecting order and stepsize; and asking over and over again, "How could I use the kind of information that the program has available or could generate, to get it to do what I know it should for this well understood problem at this particular point in the computation?" Of course, one looks for answers that are applicable to more than the simple case being examined. When unexpected behavior is encountered, ask, "Why did that happen?" Because of these efforts, my programs can do a better job of selecting order and stepsize than I can!

## 2. Rigorous selection of stepsize and order is not a piece of cake

Perhaps the most important area where theory could help is in the selection of stepsize and order. But for multistep methods reality is more complicated than the theory constructed to address these issues. When solving quadrature problems, existing theory adequately addresses the issues. But with quadrature problems the usual problems with instability do not arise.

It is useful to consider the simple test problem $y' = \lambda y$ for constant complex $\lambda$. Although this simple problem does not capture the complexity of real problems, it seems to capture enough to be useful; and it illustrates the problems with the theoretical approaches. I assume the stepsize, $h$, is constant, even while talking out of the other side of my mouth about changing it. As an algorist, and as one who has spent much time looking at difference tables verifying that this assumption provides useful insight, I have two (weak) excuses for such behavior. Perhaps you mathematicians could do better (that is, either in behavior or excuses).

For the problem above, a multistep method has a principal root, $r_p$, and $c_p r_p^n$ approximates the solution with order $q$, where $q$ is the order of the method and $c_p$ is a constant. Unfortunately, it also has extraneous roots which are introduced as a result of the difference equation having a higher order than the differential equation. For all practical methods I know of, the largest of these extraneous

roots, $r_e$, near the stability boundary for a method has a negative real part. Accuracy considerations on the other hand, limit the argument of $r_p$ to (approx.) $< 30^\circ$, corresponding to 12 samples per period.

Since the k$^{\text{th}}$ backward difference of $r^n$ is $r^n(1-1/r)^k$, both error estimation and order selection depend on expressions approximated by

$$c_p r_p^n (1-1/r_p)^k + c_e r_e^n (1-1/r_e)^k.$$

Because of their arguments, the contribution of $r_p$ to the $k^{\text{th}}$ difference decreases with increasing $k$, while the opposite is true for the contribution of $r_e$. Since, $|c_e| < |c_p|$ (if there is any relative accuracy), the magnitude of the differences decreases at first, but ultimately increases. A variable order method selects the order very near this turning point. Thus the quantities used for decision making depend on both $r_p$ and $r_e$, while the standard theory only applies to $r_p$. Although $(1-1/r_p)^k$ is proportional to $h^k$, $|1-1/r_e|^k$ will actually decrease with increasing $h$ since $\Re(r_e) < 0$, and $|r_e|$ increases with increasing $h$. Quite quickly, this reduction in $|1-1/r_e|^k$ is more than compensated for by the increase in $|r_e|^n$. Of course, if the lack of convergence in the differences is due to round off error, an increase in $h$ will lead to a smaller than expected increase in the error, particularly if this larger stepsize results in an increase in the order. Except for the comment on round off error, all said above about increasing $h$, also applies to increasing the order. That is, an increase in the order tends to make $|r_e|$ larger.

Of course changing stepsize or order will in effect change the value of $c_e$ as well as the value of $r_e$. More important, the complex numbers can't be examined directly. In a system of real equations, the complex quantities all get combined in such a way that the result is real. Imagine for example trying to determine if $|\rho^k e^{ik\theta}|$ is an increasing function of $k$, given only $\rho^k \cos k\theta$, for a few values of $k$. More than one $r_p$ and/or $r_e$ complicates matters even more. That is, if one is still interested in trying to treat this problem rigorously.

I have in fact observed differences not increasing as rapidly as expected after a stepsize increase. And sometimes a questionable order increase is followed by a clearly undesirable one. The cautious will claim this shows one should not allow stepsize or order to change too frequently. But with such prudence, one pays a price in performance when a problem should have just such rapid changes. One is probably better off overall to make small mistakes in stepsize or order selection, and then to correct mistakes as quickly as possible.

The integration order is selected in such a way that (usually) $|r_e| < |r_p|$ and thus the perturbations in the solution due to $r_e$ tend to damp out. The important point is that **error estimates do not behave in the same way as the actual local errors that matter.**

Table 1
Features in DIVA

| |
|---|
| G-Stops – Locating zeros of arbitrary functions of the solution. |
| Direct integration of second order systems. |
| Saving the solution for later use. |
| Internal diagnostic print. |
| Support for integration of variational equations. |
| Flexibility in specifying output points. |
| Reverse communication is supported, but not required. |
| Flexible error control. |
| User can control stepsize and order. |
| Diagnoses unreasonable accuracy requests. |
| Efficient integration over weak discontinuities. |

## 3. Features from the past

Table 1 lists features in a variable order Adams code, DIVA [13]. This code is essentially the same as a code finished in 1975, except for minor changes in the algorithms and the addition of features requested since then. The main ideas in the algorithms are in [10]. A more accessible reference is [17], which, however, does not include details for higher order equations or for implementing the backward differentiation formulas, and does not give the most efficient algorithm for computing interpolation coefficients.

Jackson [4] examines a variety of methods proposed for computing with divided differences or variants thereof, and concludes the algorithms in [10] are best. Jackson and Sedgwick [5] give a framework for analyzing different ways of implementing Adams methods. They conclude that *scaled* divided differences are most efficient for "small" systems, and the Lagrange form for "large" systems. ("Small" gets bigger for higher order systems, or if different integration orders are being used on different equations.) They use "scaled" divided differences for what I have called "modified" divided differences. The less descriptive term "modified" came from a paper by Blanch [1], which in fact uses a different definition. I plan to use the more descriptive term "scaled" in the future. Shilling [18] concludes that for nonstiff problems, the method in [10] is clearly superior to the Nordsieck form, with the two being about equal in the stiff case.

Allowing the direct integration of second order systems costs very little in the code. The procedure for computing integration coefficients for a first order system computes in the process, coefficients for integrating higher order systems. Since one only needs to maintain half the number of difference tables, the integration overhead per equation is cut by a factor of almost 2. In the case of implicit methods, one also saves on the linear algebra cost. There are important classes of problems [11] where the direct integration of a second order system requires about half the number of function evaluations required for integrating the equivalent

first order system. It should be noted that if the first derivative of the solution is present in the expressions for the second derivatives, it is possible for the direct integration to require more function evaluations. Even in such cases, the reduction in integration overhead is likely to make up for any extra function evaluations required. The error in second order systems is controlled by allowing the user to specify an error tolerance only on the first derivative of the solution. This has worked well in practice, and eliminates the possibility of inconsistent accuracy requests on position and velocity. **However, I know of no theoretical results indicating the best strategy.**

Diagnostic and debugging print are intended to aid in the identification of problems in the algorithm or errors in the user's problem formulation. Instead of telling the user (as in the distant past) to print this and that, to track down problems, the user is told to turn on this kind of output, at that level. From experience with the former approach I know one gets poorly labeled output, badly organized on the page, with not quite enough digits on certain key variables, and some desired data missing!

The support for variational equations uses a method of the form $PE_1C_1E_2C_2E$, where subscripts indicate the action is to apply only to the first or second set of equations, and P, E, and C are used to denote Predicting, Evaluating the derivatives and Correcting respectively. Letters without subscripts imply the action is to apply to both sets. Values of the solution for variables in the second set have no influence on the derivatives of those in the first set. Thus in the case of variational equations, the partials matrix need only be computed once per step, such values being computed from the corrected values for the state. This feature is quite easy to implement, and has a big payoff for some problems.

In [10] a method is described for detecting computational noise. This test does not make assumptions about the precision with which derivatives are computed, and as a result has found the following kinds of errors:

1. User thought all of his code was in double precision, when in fact certain computations were done in single.
2. User thought computing attraction of sun and earth in double precision, while computing the rest in single precision (to save time and space) would be sufficient to get the desired accuracy. It usually was – unless the spacecraft got a little too close to the moon.
3. User did not realize computing with time based at the time of Caesar caused cancellation which made the desired accuracy impossible to obtain.

Tests based on the precision of the floating point arithmetic have no chance of catching such errors, and these problems are of a nature where they could easily go undetected. The test is based on examining the rate with which differences get smaller, together with the place where the decrease ends. (If $r_p$ is so close to 1 that differences start decreasing rapidly, then $h$ must be close to 0, and thus $r_e$

must be small. If $r_e$ is small then lack of convergence must be due to noise.) As given in [10], this test can give a false positive result, and so has been modified to form a second difference of derivative values for very closely spaced values of the independent variable. The diagnostic is only given when this difference is sufficiently large. These two extra derivative evaluations are requested when the test in [10] indicates problems, which happens very rarely. As part of the testing process, the step is made easier to increase when it looks like noise might be limiting accuracy, and thus the overall effect of the test is to make the code more efficient when noise is (nearly) limiting precision. **Perhaps some test for noise could be placed on some kind of theoretical foundation?** In the past I have not tried to estimate Lipschitz constants which in turn could be used to estimate a bound on the eigenvalues. Such estimates might prove useful both here and in selecting stepsize and/or order.

When saving the solution, restarts require more storage to save the solution because the low order of the difference tables requires saving information more frequently. This became a problem when integrating trajectories for earth satellites which pass in and out of the Earth's shadow. It is easy to imagine such concerns also arising in connection with integrating delay equations. Simply ignoring discontinuities is very unreliable if one is using my code, while the infamous John Butcher assures me that the one step code STRIDE is reliable in this case. Thus, either the unreliability is common to multistep methods, rooted in attempting to use invalid past information, or is due to DIVA (for good reasons) not allowing the order to drop by more than one on a single step. But even if ignoring discontinuities were reliable, doing so is still quite inefficient.

I have investigated two approaches to integrating over weak discontinuities. Both require the user to identify the amount and the point of discontinuity, and both assume information in the differences prior to the discontinuity are of some use in predicting the differences after the discontinuity. (The "G-Stop" feature built into the integrator makes it easy for the user to identify the location of the discontinuity.) The first approach involves doing a Picard-like starting procedure using the new definition for the derivatives prior to the point of the discontinuity, and starting with the same values for all of the differences except for the $0^{\text{th}}$. This procedure worked well for small discontinuities, but performance degraded fairly rapidly with increasing size of the discontinuity. More important, there are cases where it is not possible to extend the new derivative evaluation to points prior to the discontinuity. Thus my users had to do something quite special as the spacecraft moved from full sun or shadow into partial shadow. (There is no problem extending the derivatives when moving from partial shadow, as full sun or shadow is easy to model; extending partial shadow is impossible.)

The second approach is more in the spirit of variable order methods. Upon being informed of the discontinuity and its size, DIVA reduces the integration order to the point where the largest order difference is larger than the size of

the discontinuity, and adjusts the stepsize accordingly. A little special logic is required to get the solution on the first step past the discontinuity. Then for a number of steps depending on the integration order, DIVA is extra careful in terms of being a little more conservative in estimating errors, and being willing to reduce the order more quickly. Overall the two approaches were about equally good, although this last approach is distinctly better for large discontinuities. Using ideas similar to those described later for starting the integration should make this second approach work even better.

It should perhaps be emphasized that one need not reduce the order of the method to the order of the discontinuity. Clearly if there is a jump in the value of the derivative much smaller than the error tolerance and the fundamental nature of the problem has not changed, all the differences are likely to be useful in propagating the solution. Large jumps will typically require a reduction in order, but one can expect, differences with an order significantly greater than one to be useful on many problems even with a jump in the $0^{\text{th}}$ derivative. Of course, the user will pay a penalty in efficiency if he chooses to use this feature when the differential equation takes a completely different form after the discontinuity.

Thus frequent discontinuities do not necessarily give one step methods an advantage as is sometimes thought. And if the discontinuities are frequent enough, a variable order code running at first order would be impossible to beat!

## 4. Features for the future

Table 2 lists the features I plan to put into a new integrator. More details can be found in [14]. Other obligations will probably delay the start of programming until well into 1995.

Topics from Table 2 are discussed below. Constraints on the size of this paper limit the detail with which these topics can be discussed.

Various aspects of how I plan to use BDFs for stiff equations can be found in [15], [16], and [20]. Unlike most implementations using BDFs, an iteration is not used in getting a solution to the corrector formula. Rather, the method is semi-implicit using some fixed number of iterations (usually 2, sometimes 1). The method uses the true variable coefficient value in computing the residual for the equation to be solved. It is not true, as is frequently thought, that such a method must factor the iteration matrix frequently. Results given in [20] certainly suggest that this general approach is promising.

The restriction to index 1 on the DAE's is not expected to unduly constrain the range of applications, while greatly simplifying the code development. For higher index problems, the user will need to reduce the index by differentiating (some of) the equations, and solve a system consisting of differentiated equations together with equations which have not been differentiated, subject to constraints

Table 2
Features planned for a new integrator.

| |
|---|
| Everything in Table 1 |
| Stiff Equations – plan to used backward differentiation formulas, BDFs. |
| Differential Algebraic Systems, DAE's (index $\leq 1$). |
| Delay Equations. |
| Systems subject to constraints. |
| Support for changing the independent variable. |
| Support for using extra derivatives. |
| An improved starting procedure. |
| A simplified error control. |
| Provision for partitioning equations to use different methods. |
| Automatic selection of method, Adams or BDFs. |
| Provision for carrying additional precision in independent and dependent variables. |
| Continuous order selection? |
| Provision to get a solution depending continuously on the input parameters? |
| Provision for different stepsizes on different equations? |
| A new user interface |

based on the differentiated equations prior to being differentiated. The feature for imposing constraints on the solution is used to satisfy these constraints. The user could of course use this policy to reduce the index to 0, but in this case the payoff is substantially less, since the BDF method works quite satisfactorily in this case.

At the request of a user, a modification was made to my current integrator to solve a delay equation. This problem involved delays from tidal forces propagating through the moon, with the state to the left of the initial point to be determined in such a way as to give a smooth solution. This initial state was determined by initially assuming the values at the delayed time were the same as at the current values, and doing a Picard-like iteration to get this initial state. Since many physical processes are characterized by smooth solutions, it is reasonable to expect this kind of problem is common. That is, delay problems where the state prior to the initial point is defined only by the requirement that the solution be smooth at the initial point. It appears awkward to treat such problems with one step methods as they don't have the equivalent of a difference table to check on the smoothness of the solution.

I first suggested modifying the numerical solution of a differential equation to comply with constraints in [8], and a number of others have independently come up with this idea. The basic idea is to project onto the constraint just after correcting and just before computing the last derivative evaluation of the step. The projection involves one iteration of solving an underdetermined system in the linearized constraints. Thus, the cost is quite low. With just a single constraint, the cost can be made even lower by moving onto the constraint in the direc-

tion of the difference between the predicted and corrected values. **(At least this worked very well for the one problem on which it was tried!)** This latter procedure requires computing the value of the constraint after predicting and correcting, instead of requiring the value of the constraint and it's partial derivatives after correcting. This technique was tried on a two-body problem with eccentricity .5, over 21870 (lots!) of periods, constraining the energy to be constant. Without the constraint errors grow quadratically with time; with it, the error grows linearly. In comparison with the best results by Calvo and SanzSerna [2] for symplectic integrators, DIVA without constraints took less than half as many function values for a given accuracy, but could not get errors below $10^{-3}$, where [2] gives results almost down to an accuracy of $10^{-4}$. With constraints, DIVA with an error 1/100 of those reported in [2], required less than 1/10 as many function evaluations. Though this problem does not show symplectic integrators at their best (because of the moderate eccentricity), it certainly suggests that the use of constraints with an existing integrator offers better performance than symplectic integrators when such a procedure is feasible. For those interested in making comparisons, results with the constraint are given for DIVA on this problem in the form (function evaluations, largest error through 21870 periods): $(4372756, 1.22\times 10^{-3}), (5213486, 8.38\times 10^{-5}), (6147913, 3.54\times 10^{-6}), (7106722, 1.10\times 10^{-6})$, and $(8047023, 1.00\times 10^{-7})$. These results were obtained with input tolerances of $10^{-k}$, for $k = 7, 8, ..., 11$, and $(x_0, x_0', y_0, y_0') = (.5, 0, 0, 3.)$, using IEEE 64 bit floating point.

There are many problems where a change in independent variable can make a significant difference in the efficiency of an integration. An example is given in [9], where a factor of 2 improvement is realized. For problems with very rapid changes in the solution, people at TRW were using arc length as an independent variable at least 25 years ago. But if a user wants to make such a change in variable, getting output at the desired points can become much messier to set up. I would like to encourage the user to make changes in the independent variable when it is desirable, by setting up the code to use the G-Stop feature on the solution for the original independent variable to get output at the points where it is desired. Thus the user has little to do other than defining $dt/d\tau$, where $t$ is the old and $\tau$ the new independent variable. Note that the iteration to find the G-Stop for a particular value of $t$ only requires the interpolations be done for that variable. The remaining interpolations can be done after the desired value for $\tau$ is obtained.

I experimented some with using extra derivatives in [6]. If one has an integrator set up to solve arbitrary order differential equations directly, this feature is quite easy to add. One treats the system as if it had order $d + e$, using a method such as that described in [10] for higher order equations, where $d$ is the original order and $e$ is the number of extra derivatives; except, the user-provided values for derivatives from $d$ to $d+e-1$ are used instead of the values ordinarily computed by the integrator for an equation of order $d+e$. Long ago I felt the extra complexity

was not worth the cost and thus have not had this feature in recent integrators. The expected development of convenient software for automatic differentiation, and the chance that the use of extra derivatives will frequently enable effective use of PEC mode when PECE would otherwise be preferred, has prodded me into including this feature. **The error control here will be based on the $d-1^{st}$ derivative, not on the $(d+e)-1^{st}$ derivative.**

The new approach to starting arose as a result of work on integrating across discontinuities. The failure of one "great" idea could only be explained by the extreme growth of rounding errors in the presence of a rapidly increasing stepsize. Clearly the same problem arises when starting an integration with a first order method. The problem is easy to see when working with scaled divided differences. But, please note that the use of differences is not the source of the difficulty. Assume $k$ steps in the starting process with the order increased by 1 on each step. If the order is then to be held constant (or perhaps even reduced), the usual procedure is to discard information from the most distant point, i.e. the initial point. But observe that the initial point is the most accurate information available, and that there is a great deal of cancellation in forming the difference between the initial point and the point following. If the point just after the starting point is removed, the shortest distance between two points is maximized, thus reducing as much as possible the growth of round off error.

The easiest way to implement this starting procedure is probably to keep the initial point as the base point of the integration until it is discarded. Thus after getting values at $t_{n+1}$ (the current value of the independent variable) they are inserted into the difference tables between $t_n$ and $t_0$. The process for updating the differences is a little different, but, except for the stepsize history, the procedure for computing integration coefficients remains the same. This process has the advantage of integrating from $t_0$ to $t_{n+1}$, instead of from the $t_n$ to $t_{n+1}$. Because of the way errors in the interpolating polynomial cancel, this is just as accurate, and in fact will be slightly more accurate if some of the values discarded early are not quite as accurate as they should be. During this process one need only compute one derivative value per step. When the difference between $t_0$ and the $t_k$ closest to $t_0$, is nearly as big as $|t_{n+1}-t_n|$, it is time to modify the difference tables by discarding the data at $t_0$. This is also easy to do. If a history is being saved for later interpolation of the solution, it should be saved just before discarding the data for $t_0$. Note that the usual starting procedure requires more storage for saving the solution, since it must save low order closely spaced difference tables near the starting point.

In the past I have tried to provide a way for the user to control the accuracy by means of a single parameter. **Ideally there would be a theoretical result such as: given the information accessible to a code, this is the best error control to use.** Lacking such a result, an error relative to the change in the solution over a single step looks good to me. To avoid problems with zero crossing

an exponential averaging technique is used. Thus for some $\alpha$ ($\approx 3/4$), use an error tolerance relative to $\alpha$ $(lastused) + (1-\alpha)|currentchange|$. Unfortunately, the starting stepsize at first order is abnormally small. Here one would like to use the expected value for the stepsize assuming a more realistic value for the integration order. This little problem has kept me from implementing this procedure. I'm hoping the new starting procedure described above will make this type of simplified error control feasible. The goal is not to do the best possible thing, but to do at least as well as a user is likely to do when he is uncertain about the nature of the error growth and the size of the solutions in his problem.

I have long advocated partitioning stiff equations to reduce the size of the linear systems involved in getting a solution. The simple minded approach suggested in [7] for doing automatic partitioning is probably not workable. A rough outline for a more sophisticated approach is given in [12]. But there are many cases where the person with the problem knows how equations should be grouped so that it is appropriate to integrate all of the equations in the group with the same method. Thus, for example, control equations for keeping a spacecraft oriented belong in a different group from those which model the external forces on the spacecraft. I believe software for solving difficult problems should use all available help. The user interface described in [14] attempts to encourage a user in this direction.

Some of my users are interested in getting solutions pushing the accuracy available with IEEE double precision format. One can pick up a little accuracy by maintaining the independent and dependent variables in extra precision. I plan to offer such an option. This will require hand coding extra precision addition only, which is quite simple. The basic idea goes back to Gill, [3]. (One of my first programs ($\approx$ 1960), and the only Runge-Kutta method I've written, used this method coded in absolute hex on an ALWAC III–E (512 bytes of memory). Assembly language, if even available then, didn't allow all the tricks that were desirable!)

When selecting the integration order, particularly at high order, it is easy to be at an order where everything is nicely stable, and getting further outside the region of relative stability than is desirable with the order increased by 1. This suggests one might use a method of "real" order. Let $q$ be such a real order, and let $k$ be the integer part of $q$. The method being proposed is simply the usual method of order $k$, with $(q-k)$ times the first neglected term in both the predictor and the corrector added to the usual formulas. Clearly such a method, although of order $k$, will have properties between those for the method of order $k$ and the method of order $k+1$. Another reason for considering such an algorithm is to open the possibility of getting a solution that varies continuously as a function of the initial conditions or some parameter appearing in the differential equation. This feature is useful when solving a boundary value problem when one does not want to integrate variational equations to get partials. Without this feature, one cannot get decent values for partial derivatives by using difference

approximations, since a very small perturbation in a variable can lead to selecting a different order, which in turn can lead to a different selection of stepsize, with the result that the perturbation leads to larger change in the solution that is appropriate. (Note that most one step methods do not suffer from this kind of difficulty.) I have only recently realized that getting this property when integrating a stiff equation will probably require computing the Jacobian matrix and factoring the iteration matrix on every step since otherwise decisions to do this on different steps for a different value of the perturbed variable would lead to the same kind of problems as mentioned above. One would probably be better off integrating variational equations. Thus, some of my enthusiasm for using real values of the integration order has been dampened. However, the idea seems of some interest for smoothing the order selection process, and is used in the next section of this paper in investigating "optimal" formulas.

Multirate methods are those which allow different stepsizes to be used on different equations. This option was originally specifically excluded because of the code complexity involved in including it. Since then, two parties have been interested in this feature, and I have kidded myself into believing that perhaps the implementation isn't too messy. I believe a key to making these methods work is to use some kind of averaging to feed the contribution from the small stepsize components into the derivatives computed for the large stepsize components. For example, astronomers used to lump Mercury in with the sun when solving for the positions of the outer planets. By explicitly computing an average (i.e. integral) while integrating the motion of Mercury with a small stepsize, one can improve on such an approximation. (One could integrate the accelerations due to the interactions of Mercury with the other planets, for very precise results, or simply average the position of Mercury for a cruder approximation.) Almost any approximation would be better than simply using the value one has for the position of Mercury at the end of the large steps being taken for the outer planets. Similar ideas of course apply to other problems where one might want to use a multirate method.

One aspect of the user interface is crucial. Whatever approach to a user interface you use, try to make it extensible. I have modified an existing code over a period of almost 20 years, in the process adding significant features not in the original design. Past use of the program was invalidated once, to provide better portability of saved solutions created on different computer systems. (It's hard to think of everything!) Since features could be added without invalidating existing code, one version of the program is maintained, and everybody using the code can make use of the latest version. Although the design is different, the new integrator keeps this important characteristic. In particular, no matter what kind of method is used to solve the corrector equation when using an implicit method, the same core code will be used. If one wants to use the latest sparse solver, it should just be a matter of adjusting the user interface slightly for that

code, and using it.

## 5. What is an optimal formula?

Long ago there were efforts to improve the order or the accuracy of formulas, without any consideration of stability. Then, there were efforts to improve stability with only minimal consideration of accuracy. Of course, improving one usually means damaging the other. As one wants more accuracy, it is best to use more accurate formulas, sacrificing stability to get the accuracy, and vice versa. Variable order methods provide a means to pick the nearly best compromise automatically. But this leaves open the question of just which variable order formulas to use. Skeel [19] has shown it is possible to have a great deal of freedom in the choice of formulas without sacrificing storage. The best choice may indeed lie in such formulas. I have wondered for some time about doing something less general, but significantly easier to implement.

The corrector formula for the Adams methods, applied to first order systems, can be written as: $y_{n+1} = p_{n+1} + hc(f(t_{n+1}, p_{n+1}) - p'_{n+1})$, where $y$ and $p$ denote corrected and predicted values, $c$ is a constant which depends on the method, $f$ is the derivative function, and $p'_{n+1}$ is the value predicted for the derivative by extrapolating the polynomial interpolating past derivative values at the end of the last step. Clearly there is freedom in choosing $c$, with little impact on the complexity of the method. In the past I have selected $c$ so that the corrector formula has order one greater than that of the predictor, primarily because using a value for $c$ giving the Adams Moulton formula with the same order as the predictor is not as stable on $y' = \lambda y$ for predictor orders $\leq 11$.

The issue addressed here is how to decide one value of $c$ is better than another, when a change in $c$ makes the formula more accurate and less stable or vice versa. Since order selection ultimately depends on numbers actually used in the computation, define the method to be stable if $|r_e(h\lambda)| \leq |r_p(h\lambda)|$. That is, use relative stability, relative to the principal root rather than to the true solution. If this condition is violated, then ultimately a variable order method will drop the order. The key to choosing the optimal value of $c$, is using a real valued order as introduced earlier.

With a predictor "order" of $q = k + \rho$, $0 \leq \rho < 1$, and free parameter $\alpha$, the method in backward difference form can be written

$$\begin{aligned} p_{n+1} &= y_n + h\sum_{j=0}^{k-1} \gamma_j \nabla^j y'_n + \rho\gamma_k \nabla^k y'_n \\ p'_{n+1} &= \sum_{j=0}^{k-1} \nabla^j y'_n + \rho\nabla^k y'_n \end{aligned}$$

$$\begin{aligned} y_{n+1} &= p_{n+1} + h[(1-\rho)\gamma_k + \rho\gamma_{k+1}](1+\alpha)[f(t_{n+1}, p_{n+1}) - p'_{n+1}] \\ y'_{n+1} &= f(t_{n+1}, y_{n+1}). \end{aligned}$$

where $\rho = \alpha = 0$, corresponds to the usual procedure. For $y' = \lambda y$ this leads to a polynomial of degree 2 in $h\lambda$ , and degree $k+1$ in $r$. Let $r_p$ and $r_e$ denote the principal and the extraneous root of largest magnitude as before. Consider

$$\begin{aligned} E &= \frac{|1 - e^{-h\lambda} r_p(h\lambda, q, \alpha)|^2}{|h\lambda|^2} \\ R &= |r_e(h\lambda, q, \alpha)|^2 - |r_p(h\lambda, q, \alpha)|^2, \end{aligned}$$

where E, the square of the relative error per step, is to be minimized subject to $R \leq 0$. Suppose values of $h\lambda, q$, and $\alpha$ are such that $R = 0$. If this $\alpha$ and $q$ are to be optimal for this $h\lambda$, then there must be no solution to the system

$$\begin{aligned} \frac{\partial E}{\partial \alpha}\delta\alpha + \frac{\partial E}{\partial q}\delta q &< 0 \\ \frac{\partial R}{\partial \alpha}\delta\alpha + \frac{\partial R}{\partial q}\delta q &< 0. \end{aligned}$$

Setting the determinant of this system to 0, gives the condition for an optimal formula,

$$\begin{aligned} \frac{\partial E}{\partial \alpha}\frac{\partial R}{\partial q} - \frac{\partial E}{\partial q}\frac{\partial R}{\partial \alpha} &= 0 \\ R &= 0. \end{aligned}$$

I have written a code to solve this problem for Adams formulas of arbitrary order. This project has been set aside for now, but in trying to solve this system near $q = 6$, for given values of $h\lambda$, and unknowns $\alpha$ and $q$, there were either bugs in my code or it doesn't much matter how $\alpha$ is selected.

Thanks are due the referees for comments that lead to clarifications in the presentation.

## References

[1] Gertrude Blanch, On Modified Divided Differences I, Math. Comp. 8 (1954) 1-11.
[2] M.P. Calvo and J.M. Sanz-Serna, The development of variable-step symplectic integrators, with application to the two-body problem, SIAM J. Sci. Comput. 14 (1993) 936-952.
[3] S.A. Gill, A process for the step-by-step integration of differential equations in an automatic digital computing machine, Proc. Cambridge Philos. Soc. 47 (1951) 96-108.
[4] L.W. Jackson, The Computation of Coefficients of Variable Step Adams Methods (University of Toronto, Dept. Comp. Science Technical Report No. 94, June 1976).

[5] L.W. Jackson and A.E. Sedgwick, Vandermonde Matrices, Co-ordinate Transformations, and Adams' Method (University of Alberta, Dept. Comp. Science Report No. TR77-5 September, 1977).
[6] F.T. Krogh, A Variable Step Variable Order Multistep Method for the Numerical Solution of Ordinary Differential Equations (TRW Systems Group, Redondo Beach, CA, Report No. 99900-6229-R000, May 1967). (An effort has been made to cite some of my more obscure work, for those who otherwise wouldn't know of it. All such papers cited here are available on request.)
[7] F.T. Krogh, The Numerical Integration of Stiff Differential Equations (TRW Systems Group, Redondo Beach, CA, Report No. 99900-6573-R000, March 1968)
[8] F.T. Krogh, An Integrator Design (Jet Propulsion Laboratory, Pasadena, CA, Technical Memorandum 33-479, May 1971).
[9] F.T. Krogh, Algorithms for Changing the Step Size, SIAM J. Numer. Anal. 10 (1973) 949-965.
[10] F.T. Krogh, Changing Stepsize in the Integration of Differential Equations Using Modified Divided Differences, *Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations*, October 1972, pp. 22-71. (Lecture Notes in Mathematics, Vol. 362, Springer-Verlag Berlin, 1974).
[11] F.T. Krogh, Summary of Test Results with Variants of a Variable Order Adams Method, pp. 277-281 of *Numerical Methods for Differential Systems*, by L. Lapidus and W.E. Schiesser (Academic Press, New York, 1976).
[12] F.T. Krogh, Notes on Partitioning in the Solution of Stiff Systems, *Proceedings International Conference on Stiff Computation*, Vol. II, April 12-14, 1982 Park City Utah. (Sponsored by the U.S. Air Force Office of Scientific Research.)
[13] F.T. Krogh, DIVA/SIVA, Chapter 14.1 of MATH77, Release 4.0 (Jet Propulsion Laboratory, Pasadena, CA, JPL D-1341, Rev. C, May 1992). (The entire MATH77 package is available for about $1000 from COSMIC, The University of Georgia, 382 East Broad St., Athens, GA 30602.)
[14] F.T. Krogh, Design of a New General Purpose code, DIVI, for Solving Initial Value Problems in Ordinary Differential Equations (Jet Propulsion Laboratory Section 372, Internal Computing Memorandum 545, Pasadena, CA 91109, April 1992). (Available via anonymous ftp, at "math.jpl.nasa.gov" in "pub/divi".)
[15] F.T. Krogh and Kris Stewart, Implementation of Variable Step BDF Methods for Stiff ODE's, in *Numerical Methods for Solving Stiff Initial Value Problems*, Proceedings, Oberwolfach, 28.6 - 4.7.1981, Edited by Germund Dahlquist and Rolf Jeltsch, August 1981.
[16] F.T. Krogh and Kris Stewart, Asymptotic ($h \to \infty$) Absolute Stability for BDFs Applied to Stiff Differential Equations, ACM Trans. on Math. Soft. 10 (1984) 45-57.
[17] L.F. Shampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations, The Initial Value Problem* (W.H. Freeman and Co., San Francisco 1975).
[18] John Shilling, A Comparison of the Nordsieck and Modified Divided Difference Storage Methods in the Computer Solution of ODEs (Univ. of Illinois at Urbana-Champaign, Dept. of Comp. Science Report No. UIUCDCS-R-83-1131 May 1983).
[19] R.D. Skeel, Construction of Variable-Stepsize Multistep Formulas, Math. of Comp. 47 (1986) 503-510 and S45-S52.
[20] Kris Stewart, *Semi-Implicit Backward Differentiation Formulas* (PhD Dissertation, The University of New Mexico, Albuquerque, April 1987).