

All Mary Sue
for directions
about statement

An Adams Guy Does the Runge-Kutta

Fred T. Krogh

Jet Propulsion Laboratory, Pasadena, CA

An 8th order explicit Runge-Kutta code, DXRK8, based on formulas of Dormand and Prince has been developed. It provides a wide variety of options and incorporates some (minor) new algorithms. Based on the testing presented dohe, the code compares well with other Runge-Kutta codes. Sufficient results are given for the reader to judge for themselves how this code compares with a variable order Adams code of the author's. The author remains a fan of multistep methods.

Categories and Subject Descriptors: G.4 [Mathematical Software]: Certification and testing; G.1.7 [Numerical Analysis]: Ordinary Differential Equations—Single step methods

General Terms: Algorithms, Languages, Performance

Additional Key Words and Phrases: Runge-Kutta, Fortran, C, Testing

1. INTRODUCTION

When asked to write a *new* Runge-Kutta code I began by trying to convince the requester that for his needs, precise tracking of GPS satellites, a variable order Adams code would be more appropriate. Among the requirements, all of which are reasonable to expect in a general purpose library code, were the following:

Output at Arbitrary Points.

G-Stops. That is, the capability to give output (and perhaps change the definition of the derivatives) at points defined by a function of the solution. A vector of such functions can be defined by the user.

Extrapolator G-Stops. As for G-Stops above, but with the constraint that no derivative is evaluated beyond the point defined by the G function. This is desirable (it can be fudged) in the model for solar pressure. There is no problem in using the interpolator G-Stops when passing from full sunlight to partial shadow (the earth's penumbra) or when passing from full shadow to partial. But there is no reasonable definition for computing the derivative when leaving the penumbra.

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Author's Address: Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91 109; email: fkrogh@math.jpl.nasa.gov.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept, ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org. © (If published by ACM) 1997 by the Association for Computing Machinery, Inc.

Mary Sue -
Do you have any comments
about the above?

An Adams Guy Does the Runge-Kutta

Section **395**
computing Memorandum 554

March 20, 1997

Fred T. Krogh

California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA **91109**

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

An Adams Guy Does the Runge-Kutta

Fred T. Krogh

Jet Propulsion Laboratory, Pasadena, CA

An 8th order explicit Runge-Kutta code, DXRK8, based on formulas of Dormand and Prince has been developed. It provides a wide variety of options and incorporates some (minor) new algorithms. Based on the testing presented, the code compares well with other Runge-Kutta codes. Sufficient results are given for the reader to judge for themselves how this code compares with a variable order Adams code of the author's. The author remains a fan of multistep methods.

Categories and Subject Descriptors: G.4 [Mathematical Software]: *Certification and testing*; G.1.7 [Numerical Analysis]: Ordinary Differential Equation--Single *step methods*

General Terms: Algorithms, Languages, Performance

Additional Key Words and Phrases: Runge-Kutta, Fortran, C, Testing

1. INTRODUCTION

When asked to write a new Runge-Kutta code I began by trying to convince the requester that for his needs, precise tracking of GPS satellites, a variable order Adams code would be more appropriate. Among the requirements, all of which are reasonable to expect in a general purpose library code, were the following:

Output at Arbitrary Points

G-Stops. That is, the capability to give output (and perhaps change the definition of the derivatives) at points defined by a function of the solution. A vector of such functions can be defined by the user.

Extrapolatory G-Stops. As for G-Stops above, but with the constraint that no derivative is evaluated beyond the point defined by the G function. This is desirable (it can be fudged) in the model for solar pressure. There is no problem in using the interpolator G-Stops when passing from full sunlight to partial shadow (the earth's penumbra) or when passing from full shadow to partial. But there is no reasonable definition for computing the derivative when leaving the penumbra.

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration, Author's Address: Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91 109; email: fkrogh@math.jpl.nasa.gov.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept, ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© (If published by ACM) 1997 by the Association, for Computing Machinery, Inc.

First derivative terms may be present. Diag is probably not going to be in the model, but just in case the code should be general enough to handle it. This precluded the use of Runge-Kutta-Nystrom formulas, which otherwise would have given better performance.

No usc: of internal saved variables or common. This is to allow the integration package to switch between different satellites with all the data needed to continue the integration left in the users data space.

Allow reverse communication for computing derivatives. This means the derivatives are computed by doing a return, rather than by calling a user supplied subroutine. The code can be used with either forward or reverse communication.

The code should be reasonably efficient. The requester was not happy with the performance he was getting from a simple 4th Runge-Kutta code he had thrown together.

The code should be simple to understand. This was the primary reason a Runge-Kutta method was preferred over the use of a variable order Adams code. The interaction of some of the requirements above, with choices that were made early in the development have not led to a code which is entirely satisfactory with respect to this requirement.

Listed below are advantages that in some applications may be important.

- (1) This code provides a full set of features, and appears to compare well with existing Runge-Kutta codes. (It does not however have provision for saving the solution for later interpolations.)
- (2) Runge-Kutta codes have much better round-off characteristics than Adams codes, and thus if you are trying to get the last few bits of accuracy possible using standard floating point arithmetic, Runge-Kutta codes are the thing to use. There is a modification of the Adams codes which the author believes would give them the edge here. But it would require another derivative evaluation per step (Adams would still be more efficient.) and significant additional complexity and overhead.
- (3) The overhead is significantly lower than for the Adams code compared with here. I believe the overhead on the Adams code could be significantly improved with very careful attention to coding the loop which updates differences and predicts for the next step. The idea is to code this loop so the compiler generates code which works well in a pipelined environment. But Runge-Kutta methods should still retain a noticeable edge.
- (4) Variable order Adams methods (if they are going to be efficient) are always computing on the edge of the boundary of relative stability. As a consequence the local error estimates tend to bounce around a great deal. I have been impressed with the smoothness in the error estimates generated by the Runge-Kutta method, and this translates into slightly better proportionality between the actual errors and errors requested.

This paper begins by discussing the basic algorithm, then describes various aspects of the algorithms used in the code, gives some comments on the user interface, and finally some results. The algorithmic details are given to provide reasonably

complete documentation for what *was* done and perhaps as a source of ideas to others; I don't feel enough of the algorithmic space has been examined here to treat the details as recommendations.

2. THE BASIC ALGORITHM

Runge-Kutta formulas are used to solve

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) \text{ given,} \quad (1)$$

where \mathbf{y} and \mathbf{f} are vectors with m components. Because of the desire for efficiency the highest order formulas available which allowed for interpolation to arbitrary points, and provided for stepsize control were selected. The formulas by Dormand and Prince as used in the code DOP853 given in [Hairer et al. 1993] appear to fit this requirement. The code DOP853 was used as a starting point, although the codes now look very different.

DOP853 allows the user to interpolate to arbitrary points, or not. The interpolation option adds 25% to the number of derivatives required. Since this cost is only required on the steps in which an interpolation is actually done, it was easy to decide to modify DOP853 so that this cost only occurred on such steps. Wow, an easy 25% improvement when the steps requiring interpolation are only a small fraction of the total! The interaction of this choice with the G-Stop code and the option for using reverse communication had a greater impact than was first appreciated. The choice made was not necessarily wrong, but it was not obviously the right one. With an Adams code allowing for reverse communication, and for both types of G-Stops does not complicate the code unnecessarily, since the code is ready to do interpolation at any time. With a Runge-Kutta code one could make a good case for giving up this 25%, or not allowing for reverse communication, which would also simplify other parts of the code. Using a Runge-Kutta code with a lower order which does not require extra derivatives to do the interpolation would add much more than 25% to the cost as we shall see later.

To clarify the problem, imagine that the code has just discovered there is a G-Stop occurring inside the step just about completed. It now needs to get three more derivative values before it has the data required to iterate for the zero. With forward communication, the interpolation routine could be called, it could check if derivatives are needed, and if so compute them and store the required data. With reverse communication, one gets the code's for setup of interpolation tangled up with the code used for the various stages of the basic Runge-Kutta method.

3. ON THE CHOICE OF NORM FOR ERROR CONTROL

I have in the past used an L_∞ norm because it seemed to me not to make much difference what norm was used, and the L_∞ norm allows one to flag the offending equation when the requested accuracy cannot be obtained for some reason or there is some sign of a discontinuity. I am indebted to George Hall who suggested to me in a conversation some years ago that the L_2 norm has a desirable smoothness property. (See also, [Hall and Higham 1988].) This I believe is sufficient reason to

prefer the L_2 norm to the alternatives. Thus $\|\mathbf{v}\|$ and $\|\mathbf{v}\|_7$ are used for

$$\|\mathbf{v}\|^2 = \sum_{i=1}^m v_i^2, \quad \text{and} \quad \|\mathbf{v}\|_7^2 = \sum_{i=1}^m (v_i/\tau_i)^2 \quad (2)$$

where τ_i is the absolute accuracy currently being requested for the i^{th} component of \mathbf{y} . These formulas have been written in terms of the squares to emphasize the fact that the code internally works with the squares of norms most of the time. If there were a direct estimate e_i for the error in y_i (which it turns out there isn't) we would be interested in keeping $\|\mathbf{e}\|_7$ somewhat less than 1.

4. GETTING THE STARTING STEPSIZE

I recommend [Gladwell et al. 1987] and [Watts 1983] as starting points for information on what has been done by others. My biases have led to different choices than made in these papers. Most important, if one changes the problem by replacing y with αy , t with βt , changes the τ_i by a factor of α , and changes all t output points by a factor of β , it is desirable that (to within the effect of round-off errors) one get α times the solution obtained with $\alpha = 1$. If any part of an algorithm does not have this kind of scale invariance, it will certainly make the wrong choices in some situations.

Secondly I wanted something simple, with assumptions that are easy to understand. The former because one can not do a perfect job at reasonable cost, so why not do something cheap that works well a good part of the time? The second so the user has some chance of recognizing when the automatic choice is not likely to do a good job or more likely to recognize why a bad automatic choice is being made. It should also be kept in mind that for many problems the user will have a better idea of what to use for an initial stepsize than does the code.

The algorithm sketched here requires one additional function evaluation. One might compare this with the variable order Adams code DIVA used in some comparisons later. DIVA requires the user to make a guess, but even with very bad guesses it rarely requires more than two additional function evaluations.

It is assumed the norm of the $k+1^{\text{st}}$ derivative of y is ρ times the norm of the k^{th} derivative, independent of k . Norms are weighted L_2 norms based on the accuracy requested for the various components of y . Clearly this is not an assumption that one would want to count on, but it does have the advantage of scale invariance. One should keep in mind that if a poor choice is made, it probably only costs at most a couple of steps worth of function evaluations, and if the choice is so terrible that it leads to some arithmetic exception, one can input a starting stepsize. This assumption implies that the starting stepsize should be proportional to $1/\rho$ since the error should be proportional to $(h\rho)^8$.

The extra derivative evaluation is obtained after taking an Euler step of length

$$|h| = .0625 \frac{\|\mathbf{y}\|_7}{\|\mathbf{f}\|_7} (d/\|\mathbf{y}\|_7^2)^{1/16} \quad (3)$$

where the square of the ratio of the estimated to requested error on later steps is not allowed to exceed d , and $|h|$ is set to .0625 if $\|\mathbf{f}\|_7$ is 0. (In the code $d = .16$ is the default, which means a step with estimated error / requested error $> .4$ is

reject cd.) The constant .0625 was picked as it seemed to work reasonably well, and the (1/16) since the method is of 8^h order (recall it is the square of the norm that is computed).

Let \mathbf{f}_1 denote the \mathbf{f} computed from the result of taking the above Euler step, \mathbf{f} , the \mathbf{f} at the initial point, and h_1 the h used in taking this Euler step. The size of the step used to take the first step is given by

$$|h_1| \min \left(\begin{array}{l} 100 |h_1| \\ 8 |h_1| / \|\mathbf{f}_1 - \mathbf{f}_0\|_\tau \\ (\|\mathbf{y}_0\|_\tau / \|\mathbf{f}_1\|_\tau) (d / \|\mathbf{y}_0\|_\tau^2)^{1/16} \\ |h_1| \|\mathbf{f}_0\|_\tau / \|\mathbf{f}_1 - \mathbf{f}_0\|_\tau \end{array} \right), \quad \begin{array}{l} \text{if } \|\mathbf{f}_1 - \mathbf{f}_0\|_\tau = 0 \\ \text{else if } \|\mathbf{f}_0\|_\tau \ \& \ \|\mathbf{y}_0\|_\tau = 0 \\ \text{else if } \|\mathbf{f}_0\|_\tau = 0 \\ \text{else if } \|\mathbf{y}_0\|_\tau = 0 \end{array} \quad (4)$$

$$|h_1| \min \left(550 \sqrt{\frac{h \|\mathbf{f}_1\|_\tau \|\mathbf{f}_0\|_\tau}{\|\mathbf{f}_1 - \mathbf{f}_0\|_\tau \|\mathbf{y}_0\|_\tau}}, \frac{3.5 \|\mathbf{f}_0\|_\tau}{\|\mathbf{f}_1 - \mathbf{f}_0\|_\tau} \right) \quad \text{Usual case.}$$

The second term in the "min" is to keep a (very crude) estimate of $|h\lambda| < 3.5$, where λ is the eigenvalue of largest magnitude of $\partial \mathbf{f} / \partial \mathbf{y}$. In the first term, the factor of 50 was picked because it seems to work reasonably well. The rest of the formula is based on taking the geometric mean of two estimates for ρ in computing the initial h . Here and elsewhere (except for the very last step), when discussing h , $|h|$ is forced to lie between minimum and maximum values which by default are 0, and the distance to the final point.

5. ESTIMATING ERRORS

An order 8 error estimate would require even more derivative evaluations than the 12 now required to take a step and thus is not available directly. As in 1101'853, an order 5 and an order 3 error estimate are available. Let $E_3 = \|\text{Order 3 estimates}\|_\tau^2$ and $E_5 = \|\text{Order 5 estimates}\|_\tau^2$. A smaller and slightly smoothed order 3 estimate (squared) is obtained with $\hat{E}_3 = .01 (E_3 + .001)$ (last value for \hat{E}_3).

If $E_5 \leq \hat{E}_3$ then errors seem to be reasonably rapidly converging and the E_5 error is extrapolated based on the value of \hat{E}_3 . Otherwise E_5 is used. Thus the estimate for the square of the error (or more precisely, the quantity used for controlling the stepsize) is given by

$$E = \frac{|h|^2 E_5 \min(1, E_5 / \hat{E}_3)}{\text{number of equations that have some kind of error control imposed}} \quad (5)$$

I like the idea of saving derivative values by extrapolating to get error estimates in this way.

6. SELECTING THE STEP SIZE

In my work on Adams codes I have found it useful to use past history to select the next stepsize based on what the error on the next step is *expected* to be, rather than simply using the error on the current step. Similar ideas are used here. An alternative would have been to use control theoretic techniques as described in [Gustafsson 1991], but there was not time to try both.

Recall that d is the value that E defined by Eq. (5) is not allowed to exceed. In addition, there is a parameter ℓ (default value is $4.6 \approx -\ln .01$) which is what the code attempts to keep $-\ln E$ close to. Logarithms are used since one gets smoother

predictions working with the logarithms of the errors. Subscripts are used below to denote the step number, with n denoting the current step, and thus E_n is the square of the error estimate on the current step.

First consider the case when a step has been rejected because E_n in Eq. (5) is too large. The code never reduces h by more than a factor of two, on the assumption that if it had done such an unsatisfactory job in prediction, there is a good chance it is due to a discontinuity in which case a binary search is going to be as efficient as anything. If the rejected step is due to a rapid change in behavior, that is more likely to come from the tail of the interval, which will play a smaller role when the stepsize is reduced. Thus it is reasonable to assume the error is proportional to h^8 .

$$h_{n+1} = \max \left(.5, e^{(\ln E_n + \ell)/16} \right) h_n \quad (6)$$

When the step is not rejected, a model is needed in order to select the next stepsize. (A common simple model would be assume that the error is proportional to h^8 .) An "observation" r_n is given by $\ln E_n - \ln \hat{E}_n$, where \hat{E}_n is $(h_n/h_{n-1})^{16} E_{n-1}$. Thus if r_n is 0, and E_n has the desired value, it appears the stepsize is ideal. But if this condition is satisfied on the current step, and on the previous step r_{n-1} was small, there is some reason to expect that it might be big on the next step and that performance will be improved if this is anticipated. The code attempts to model r_n with a linear function obtained using a weighted least squares fit to past values of r_k . Then h_{n+1} is selected by replacing $\ln E_n$ with $\ln E_n + \hat{r}_{n+1}$ for purposes of stepsize selection, where \hat{r}_{n+1} is what the linear model predicts for r_{n+1} when it is computed on the next step. Thus $h_{n+1} = h_n \times e^{-(\ln E_n + \ell + \hat{r}_{n+1})/16}$.

If the observation for r_k is weighted by $\omega^{(n-k)/2}$ and observations are assumed to have been processed forever, one obtains a simple algorithm for obtaining a solution to the fitting problem. (The code uses $w = 1/2$.) Assuming the model for r_k is given by $a + b(k - n + 1)$, we only need to solve for a . The normal equations defining the values of a and b have the form

$$\begin{bmatrix} \sum_{k=0}^{\infty} \omega^k & \sum_{k=0}^{\infty} -(k+1)\omega^k \\ \sum_{k=0}^{\infty} -(k+1)\omega^k & \sum_{k=0}^{\infty} (k+1)^2 \omega^k \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{\infty} r_{n-k} \omega^k \\ \sum_{k=0}^{\infty} -(k+1)r_{n-k} \omega^k \end{bmatrix} \quad (7)$$

which can be written

$$\frac{1}{(1-\omega)^3} \begin{bmatrix} (1-\omega)^2 & \omega-1 \\ \omega-1 & \omega+1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \xi_n \\ \eta_n \end{bmatrix} \quad (8)$$

where $\xi_n = w * \xi_{n-1} + r_n$ and $\eta_n = \omega \eta_{n-1} - \xi_n$. And thus

$$a = \frac{1-\omega^2}{w} \xi_n + \frac{1-\omega}{1+\omega} \eta_n \quad (9)$$

As written above, getting a only requires 3 multiplies and 3 adds since w is a constant. The code actually uses something slightly more complicated. If $r_n > 0$, ξ_n is not allowed to be bigger than $\max(2 * \xi_{n-1}, 4)$, and otherwise is not allowed to be smaller than $\min(2 * \xi_{n-1}, -4)$. The effect of the last of these conditions is that the stepsize can increase by a factor of at most 1.284 . . . if things are just starting to look easier, and if things keep looking sufficiently easier this factor of allowed increase is squared on succeeding steps.

Unfortunately, there is more. Initial values must be assigned to ξ and η , adjustments made when a step is rejected, and overly optimistic increases in h must be protected against. (Unlike some codes however, there is no restrictions on how big an increase in h is allowed, as long as a pattern of long term increases is established.) The following summarizes the details.

- (1) When starting set $\nu = -2$ and $\mu = -2$. At the end of the first step, set ν to -1 , and at the end of the second step set it to 0 . The use of ν is described below; the larger μ , the more conservative the choice of h . Also set $\zeta = -32$. The new stepsize h_{n+1} must satisfy $\ln(h_{n+1}/h_n) < -\zeta/16$. The only time ζ is likely to influence stepsize selection is when the program thinks there is a chance the stepsize should be restricted due to stability concerns, or just after a step is rejected, and thus further discussion of ζ is postponed to the following section.
- (2) When nu is < 0 , \hat{r}_{n+1} is set to 0 . There is not enough information on how to bias the choice of stepsize.
- (3) When nu is -1 (the second step), set $\xi_2 = r_2/(1 - \omega)$ and $\eta_2 = -r_2/(1 - \omega)^2$. These are the values these quantities would have if the current values had been the values for an infinite number of past steps.
- (4) When a step is rejected, set $\mu = 5$ (be conservative for a bit), reduce both ξ and η by a factor of $.001$ (lower the weight on the past, the present appears different), and set $\ln E_{n+1} = \hat{E}_n - \ln E_n - \ell$ (adjustment for the change being made in h).
- (5) When $\mu \leq 0$ the code is less conservative in choosing h_{n+1} . In this case $\hat{r}_{n+1} = \max(a, -\ell - \ln E_n + \mu)$. Note that $|h_{n+1}|$ will be larger the smaller, *i.e.* more negative, \hat{r}_{n+1} . Then μ is replaced with $\min(-.5, r_n + \ell + \ln E_n)$. Thus when E_n is smaller, this will make μ smaller, and this makes it more likely a will be used in place of something bigger than a in the future. If there has been no pattern of things getting easier to integrate, $\mu = -.5$ which means $|h|$ can be boosted by at most $e^{.5/16} = 1.03 \dots$ over the h that results from using the common simple model mentioned earlier.
- (6) When $\mu > 0$, \hat{r}_{n+1} is set to $\max(r_n, \mu)$ and μ is replaced with $\mu - 3$. This case happens only for brief periods after a step has been rejected.

7. CHECKING FOR STIFFNESS

One ordinarily expects $E_5 < \hat{E}_3$ (in Section 5) since the fifth order error estimate is typically quite a bit more accurate than is the third order error estimate. If this condition is consistently violated, there is a reasonable chance the system is stiff. (This can also probably occur when the accuracy requested is unreasonable given the precision with which the derivatives are computed. The code protects against, this, but unlike DIVA, only when the derivatives are computed to nearly the full precision of the arithmetic used.) This condition is used as a filter to restrict stepsize increases and to decide on whether to do a more expensive check for stiffness. The test has been tuned to work reasonably well on $y' = -y$ and thus will probably not do quite as well at restricting the stepsize on other stiff problems. (It should however do better than codes which make no effort to detect stiffness or to restrict the stepsize when such appears likely.)

Recall, that the stepsize is always restricted so $\ln(h_{n+1}/h_n) < -\zeta/16$. In order to get smoother selection of the stepsize when stability is limited by stability concerns, the code updates ζ from its initial value of -32 as follows.

- (1) If $E_5 \leq \hat{E}_3$, subtract 6 from ζ .
- (2) When $E_5 > \hat{E}_3$, $\zeta = (.00475 - \hat{E}_3/E_5)/(.0019 + \hat{E}_3/E_5)$. (See what I mean by tuning?) Perhaps the kind of thing done here might be improved with a judicious choice of the formulas for E_3 and E_5 without losing any important characteristics of these formulas?
- (3) If a step is rejected, ζ is set to 3.2.

If a high percentage of recent steps have $E_5 > \hat{E}_3$, there is a reasonable chance that stiffness is limiting h , and that a diagnostic should be given. The following algorithm is used.

- (1) Let κ be the current step number, $\hat{\kappa}$ be the step number the last time $E_5 > \hat{E}_3$ was observed ($\hat{\kappa} = 0$ initially), and σ be a variable set to -12 initially.
- (2) When $E_5 > \hat{E}_3$, assign new values to σ and $\hat{\kappa}$ as follows: $\sigma = .8^{\kappa - \hat{\kappa}} \sigma + \kappa$ and $\hat{\kappa} = \kappa$. If $\sigma > 3.3615\kappa - 5.2544$, then σ is set to -1.1×10^{30} to flag that an additional check is desired at the end of the step.
- (3) At the end of the step if $\sigma < -1.01 \times 10^{30}$, the ratio $s = h^2 \|\delta \mathbf{f}\|^2 / \|\delta \mathbf{y}\|^2$ is formed. If $s > 9$, a diagnostic is given and σ is set to -10^{30} which has the effect of delaying another diagnostic of this type for a large number of steps. If $s \leq 9$, σ is set to 0, and if $E_5 > \hat{E}_3$ continues to be satisfied frequently, the check that requires computing s will be repeated.

The coefficients for updating σ are such that if σ is very close to 0, then 5 successive steps with $E_5 > \hat{E}_3$ are just barely sufficient to trigger the additional check.

8. THE USER INTERFACE

The user interface is designed with the following goals in mind

- (1) It should be simple to use for simple use.
- (2) It should support a wide range of functionality.
- (3) It should be possible to add new features without any changes being required in the usage for codes using earlier versions. This was also a goal when DIVA was written in the early 1970's, and has proven to be a very good idea.
- (4) It should be possible to use a very similar interface for a multistep method I'm planning to write in the future. (This code would include provision for stiff, differential-algebraic, and delay equations, [Krogh 1992]).
- (5) It should be possible to use some of the features with no real need to be aware of the others. Hypertext documentation would make it possible to move closer to this goal.
- (6) As much as is reasonable, user's not interested in a feature should not pay a cost for features they are not using.

The calling sequence has the form

```
CALL DXRK8(TS, Y, OPT, ID AT, DAT, WORK)
```

where all arguments are arrays. TS contains t , h , and t_f , the final time, which must be specified, and Y contains the current value of y . OPT can consist of a single 0., in which case the integration is done from TS(1) to TS(3) using a default mixed absolute/relative error test with a tolerance of ϵ^{75} , where ϵ is the smallest number for which $1 + \epsilon + 1$.

IDAT is used to communicate status information to/from the user, contains dimension information about various arrays on the first call, and is used to store internal state. DAT and WORK are arrays used for internal floating point numbers. If calling the integrator to do separate integrations and one quits at the proper time, WORK need not be saved in order to continue the current integration.

For most USC, OPT, the option vector is needed to specify the use of optional features. We have been using option vectors of various flavors for 25 years, and have found them useful for offering a wide variety of options without undue effect on code efficiency while allowing for new features without impacting past usage. This array contains arguments that appear in groups. A group (starting in location 1 or just after the previous group) consists of an option index identifying the option, followed by a arguments that depend on the option index. The last option must be 0., which serves to flag the end of the options.

I am convinced that a code should do all it can to encourage the user to think of the equations as belonging to different groups. For simple problems there is just one group, and the user need say nothing special. Thus by using options one can specify equation group boundaries. Certain options can be specified that will apply starting with the start of the current group, until this option is changed for some other group of equations. Examples are error control (6 types, including none), diagnostic output to be printed or not, attempt to maintain extra precision for some variables, turn off obtaining interpolated values (when interpolating) or turn back on. DIVA does not have this later capability and a user integrating some 40,000 equations with all interest in interpolating only three of them could have used it. The arguments for making it easy to group equations are even more compelling when a code is designed to handle stiff equations.

Both DIVA and DXRK8 allow for the possibility that there may be several G-Stops of different types and multiple requests for output at specific values of t . I find it interesting that when output comes it has a tendency to come in bunches, and thus a higher probability than might be expected for lots of the above to occur in a single step. I have always found it a bit challenging to insure that this output is provided to the user in the correct order. This is an argument for making control of this output an intrinsic part of the code, rather than assuming that simply allowing the user to specify the next place where output is desired and assuming they can deal with complications in the tricky cases.

Since the G-Stops bring in a bit of extra code, a separate subroutine is used for this function. If this option is requested, the user will get flags from time to time letting him know that g 's need to be computed and this subroutine called. Relatively little code is needed in the main integrator to support this feature.

If reverse communication is not used, derivatives (and G-Stops) are computed in a user coded routine, DXRK8F, and when an output point is reached, the user coded routine DXRK8O is called. One can use forward communication for one and reverse for the other if desired. We have used fixed names primarily to avoid the

mistake users sometimes make of forgetting the external statement that is required when names are passed. It also makes the call a little more compact. We believe the extra flexibility of using arbitrary names does not buy much, when one can use different file names for different routines if desired, and if integrating many different cases in one routine, a case statement in a single routine is probably the cleanest way to do things.

9. TEST RESULTS

At the end of this paper are figures giving results for a number of Runge-Kutta codes and for DIVA. These results were obtained on a 133MHz Pentium (IEEE 64 bit arithmetic) with the Lahey F771 compiler. The test problems are those suggested in [Krogh 1973], where reasons for using these particular test cases are given. Since some of these test problems have also been used in the development of the codes I have written, the results undoubtedly have a slight bias in favor of my codes. (I, of course, do believe it is slight.)

References for the methods used in the various codes are: for DOP853 (and for the formulas used in DXRK8), [Hairer et al. 1993]; for RKSUITE, [Brankin et al. 1991] and [Kraut 1991] (these being cited with the documentation that comes with the code downloaded from netlib); and for DIVA, [Krogh 1974] and [Krogh 1994]. The test program allows for interpolation to output points when it is available, or integrating to an output point and then continuing from that point. The number of output points is not large, the following table summarizes where the codes are required to give results. ($K = 1.86264 \dots$)

Cases	Points
1, 2, 3, 7	10, 30, 50
4, 5, 8	$27l, 6\pi, 16\pi$
9	$27r, 47r, \dots 16\pi$
6	$K, 21l, \dots 28K$
10	$6.192169 \dots$
11	0, 1

In connection with the results, I would like to note the following.

The overhead is computed by dividing the total CPU time (which is repeatable to almost three figures) by the number of function evaluations. As would be expected, DIVA stands out in having much more overhead than the other codes. On some of the test problems, errors are computed at the end of every step, and since DIVA takes smaller steps, this works against DIVA slightly, but I suspect this bias is too small to measure. Surprisingly, even on these simple test problems, DIVA takes less total time than some Runge-Kutta methods on some problems, particularly at high accuracy. The codes in RKSUITE have surprisingly large overheads relative to the other Runge-Kutta codes. I was surprised that support for reverse communication and keeping all information on the state of the integration in arrays passed to DXRK8 did not have a noticeable impact on its overhead. Finally it should be noted that even in the most expensive case, the overhead for 10,000 function evaluations which is a reasonably long integration, amounts to less than a second.

The distance on the abscissa from a major tick mark to the minor tick mark on its right represents a factor of two in performance.

DOP853 with interpolation requires about 25% more function evaluations than without, as expected. It also has slightly more overhead per function evaluation when interpolating. Other than this, DOP853 and DXRK8 are almost indistinguishable.

There appears to be little reason to use the low order codes in RKSUITE since they are not noticeably more efficient at low accuracies and are a lot less efficient at high accuracies, and carry higher overhead per function evaluation.

The 7-8 pair in RKSUITE is slightly better on the linear problems which use a nearly constant stepsize than DOP853 and DXRK8. This appears to be due to the formulas rather than implementations of the methods. The reverse situation appears to be true for problem 8. Note that this code does not allow for interpolation. Except for problem 4, DIVA integrates all second order equations without breaking them up into first order systems.

For problems 9-11, results for DIVA could be improved by almost a factor of 2 by skipping the 2nd derivative evaluation of the step based on the results in [Krogh 1976].

Error tolerances were adjusted by a factor so that the global errors on problem 3 were nearly equal for the differ-cut methods. (Error proportionality seemed to be a bit more uniform on this problem than on the others. This factor was then used for all problems.)

The Runge-Kutta methods can get better accuracy than the Adams code.

Runge-Kutta methods are somewhat boring:-)

References

- BRA N KIN, R. W., GLADWELL, I., AND SHAMPINE, L. F. 1991. RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs. Math. Dept. Softreport 91-1, Southern Methodist University, Dallas, Texas.
- GLADWELL, I., SHAMPINE, L. F., AND BRANKIN, R. W. 1987. Automatic selection of the initial step size for an ODE solver. *Journal of Computational and Applied Mathematics* 18, 175-192.
- GUSTAFSSON, K. 1991. Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods. *ACM Transactions on Mathematical Software* 17, 4 (Dec.), 533-554.
- HAIRER, E., NØRSETT, S. P., AND WANNER, G. 1993. *Solving Ordinary Differential Equations I* (Second Revised ed.). Springer-Verlag, Berlin.
- HALL, G. AND HIGHAM, D. J. 1988. Analysis of stepsize selection schemes for Runge-Kutta codes. *IMA Journal of Numerical Analysis* 8, 305-310.
- KRAUT, G. 1991. A comparison of RKSUITE with Runge-Kutta codes from the IMSL, NAG and SLATEC libraries. Math. Dept. Softreport 91-6, Southern Methodist University, Dallas, Texas.
- KROGH, F. T. 1973. On testing a subroutine for the numerical integration of ordinary differential equations. *Journal of the ACM* 20, 4 (Oct.), 545-562.
- KROGH, F. T. 1974. Changing stepsize in the integration of differential equations using modified divided differences. in D. J. W. COLL, D. H. LAWRIE, AND A. H. SAMEH Eds., *Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations*, Number 32 in Lecture Notes in Mathematics, pp. 22-71. Berlin: Springer Verlag.
- KROGH, F. T. 1976. Summary of test results with variants of a variable order Adams method. In L. LAPLOUS AND W. E. SCHIESSER Eds., *Numerical Methods for Differential Systems*, pp. 277-281. New York: Academic Press.
- KROGH, F. T. 1992. Design of a new general purpose code, DIVI, for solving initial value problems in ordinary differential equations. Internal Computing Memorandum 545 (April), Jet Propulsion Laboratory.

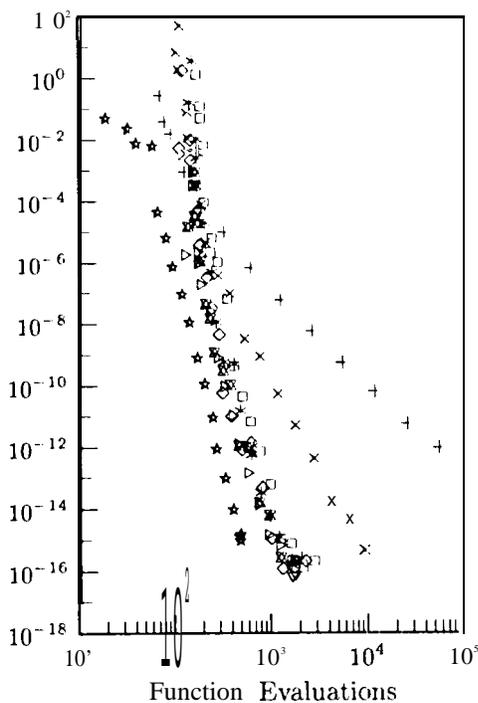
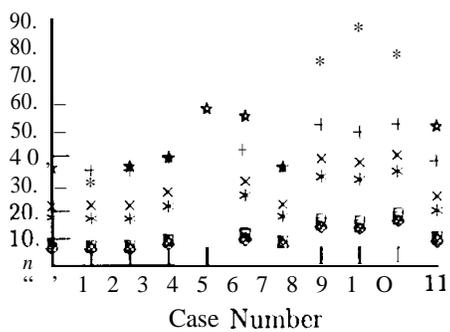
- KROGH, P. T. 1994. Issues in the design of a multistep code. *Annals of Numerical Mathematics 1*, 423-437.
- WATTS, H. A. 1983. Starting step size for an ODE solver. *Journal of Computational and Applied Mathematics 9*, 177-191.

Note, graphs are to be redone.

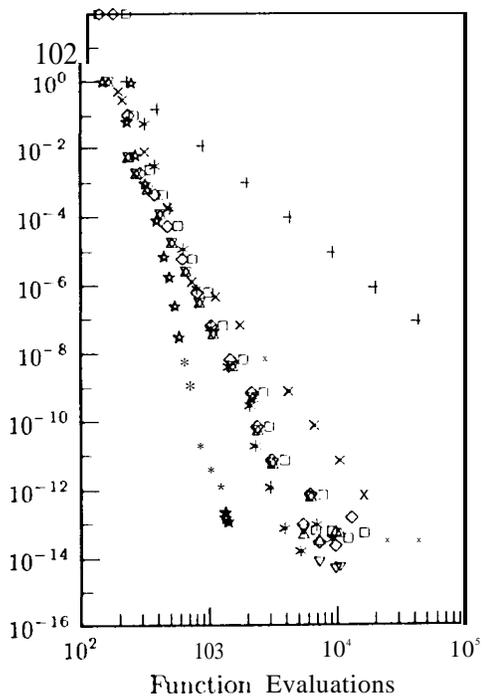
Legend

	Package	Interpolate
△	DXRK8	Yes
▷	DXRK8	No
v	DXRK8 Extra-Prec.	Yes
□	DOP853	Yes
o	DOP853	No
i	RKSUITE 2-3 Pair	Yes
×	RKSUITE 4-5 Pair	Yes
*	RKSUITE 7-8 Pair	No
*	DIVA	Yes

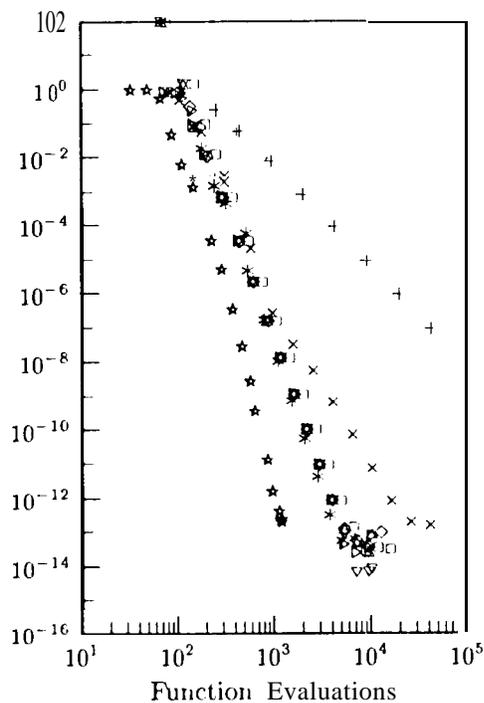
Own-head ~-Sees./F



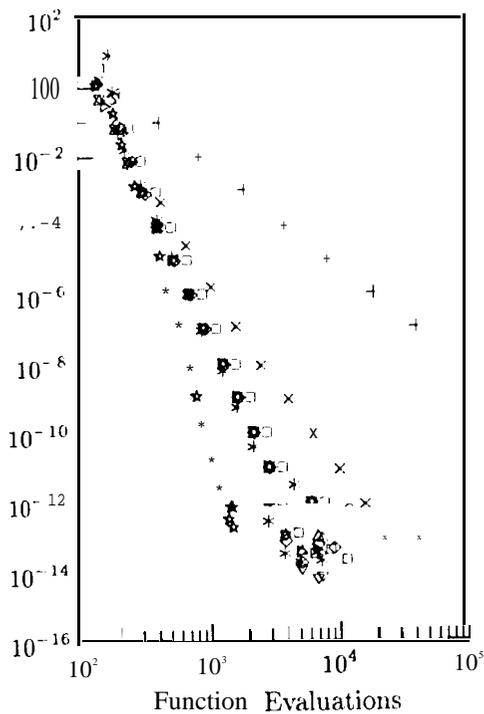
2. Relative Error for $y' = -y$



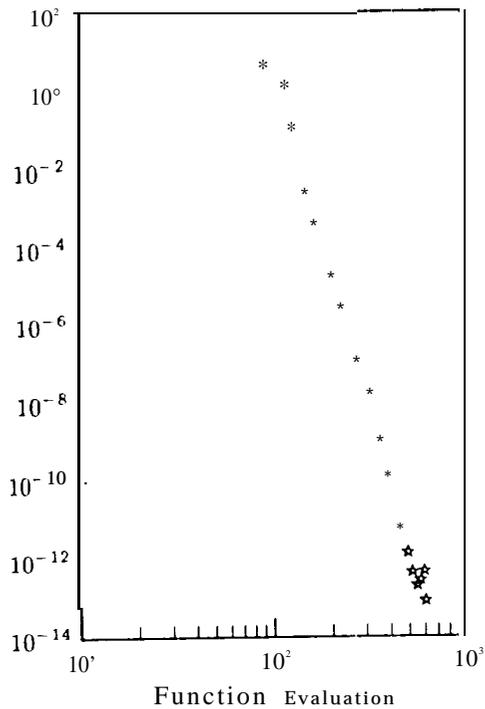
3. Relative Error for $y' = y$



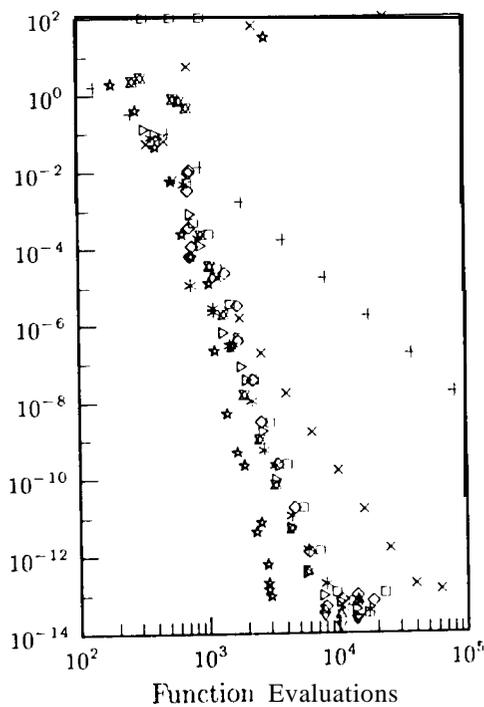
4. Absolute Error for
 $y_1' = y_2; y_2' = -y_1$



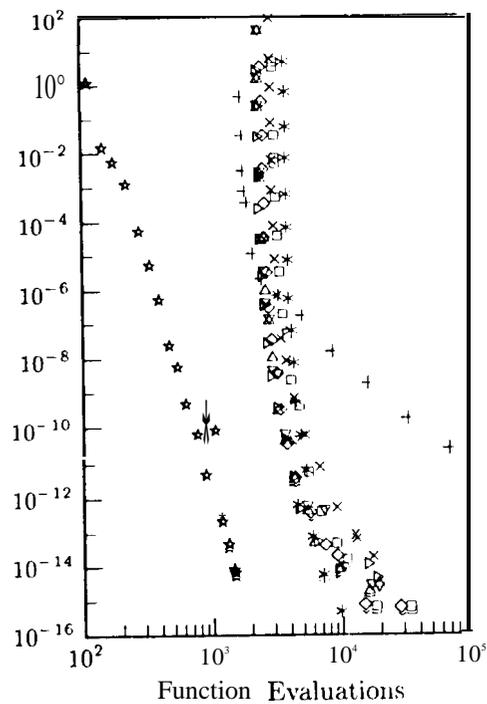
5. Absolute Error for
 $y'' = -y$ 14



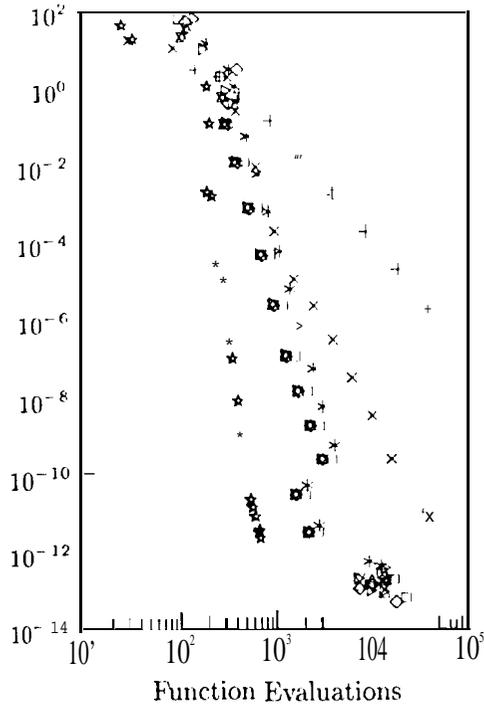
6. Absolute Error for
 Euler Equations



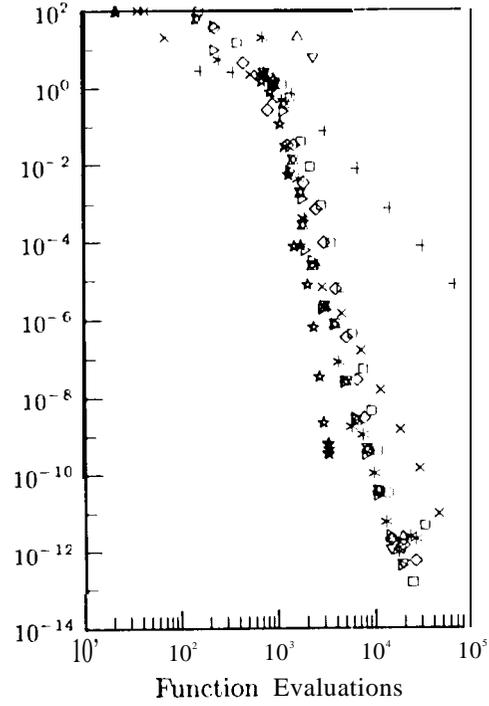
7. Absolute Error for
 $y' = t(1 - y) + (1 - t)e^{-t}$



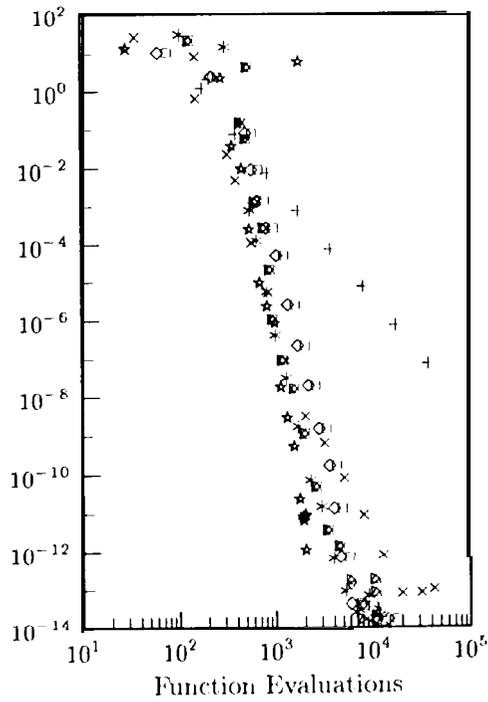
8. Absolute Error for
Circular Two Body Problem



9. Absolute Error for
TwoBody (I'01), Eccen. $\cdot 0.6^{15}$



10. Absolute Error for
Restricted 3 Body Problem



11. Absolute Error for
 $y' = (2/3) t^{-1/3}$

