UCID-16558

LAWRENCE LIVERMORE LABORATORY
University of California/Livermore, California

NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS: LECTURE NOTES

A. C. HINDMARSH

June 1974

Prepared for U. S. Atomic Energy Commission under contract no. W-7405-Eng-48

DISCLAIMER

FOREWORD

This report consists of the lecture notes for the course C703: Numerical Solution of Ordinary Differential Equations, taught in the winter quarter of 1973. The course was part of the Continuing Education Program at LLL. It met three times a week for eleven weeks, and the lectures (fifty minutes each) were videotaped. The tapes can be viewed by arrangement with the Instructional Television Center, Bldg. 131, Rm. 1267. The timetable in Appendix I gives the correlation between the topics and the lecture numbers.

The text used for the course was Numerical Initial Value Problems in Ordinary Differential Equations, by C.W. Gear. The book was not followed closely; about half of the course material was drawn from other sources. These notes are self-contained and do not require the text. However, as a convenience to readers of Gear's book, a list of errata found in it is provided in Appendix II.

The course was divided into three (unequal) parts, as are these notes. Part I is an introduction to, and a survey of, the subject. It provides a complete short course in itself, although the only methods discussed there in great detail are the Euler methods. Part II consists of a rather thorough study of Runge-Kutta methods and of linear multistep methods, with a short chapter on extrapolation. It is largely self-contained, and could be used as a short course for audiences already somewhat familiar with the subject. Part III deals with the class of stiff problems and various approaches to their solution.

The course included three guest lectures -- by Bob Pexton of Computation Department, Julius Chang of T-Division, and Ralph Willoughby of I.B.M. I am very grateful for their assistance in this course.

The notes include a few simple exercises scattered throughout, which were assigned as homework for the course.

Portions of the text have been reproduced here, by permission of Prentice-Hall, Inc., Englewood Cliffs, N.J.

CONTENTS

Part I: Survey 1

1. The problem; existence 1
2. Discretization 5
3. Basic concepts ; the explicit Euler method 6
   a) Explicit vs implicit 8
   b) Truncation error, order, convergence 8
   c) Error estimation 9
   d) Roundoff error 10
   e) Stability 16
   f) Efficiency 20
4. The implicit Euler method 21
5. Survey of method classes 28
   a) Discrete vs nondiscrete 28
   b) One-step vs multistep 28
   c) Runge-Kutta 29
   d) Linear multistep 29
   e) Extrapolation 31
   f) Splines. 31
   g) Series 32
   h) List of methods. 34
6. Analysis of one-step methods 36
   a) Stability, consistency, convergence, and order 36
   b) An example 43

7. Implementation 51

a) History vector; storage 51

b) Solution of implicit equation 52

c) Error estimation and choice of step size and order. 60

Part II: Study of Discrete Methods 72

8. Runge-Kutta methods 72

a) Explicit 2-stage 72

b) Explicit r-stage; classical case 79

c) Order, error estimation 84

d) Implicit r-stage 86

e) Higher-order methods; Fehlberg's methods; numerical examples. . 91

f) Stability 102

g) Summary 105

9. Linear multistep/multivalue methods 111

a) Basic formulas, examples 111

b) The Adams methods (explicit and implicit) 116

c) Symbolic derivations. 121

d) Order, associated polynomials, asymptotic error 131

e) Stability and convergence; root conditions. 138

f) Matrix formulations and multivalue methods. 150

10. Extrapolation methods 159

Part III: Stiff Problems. 169

11. Stiffness. 169

12. Methods for stiff problems 177

a) A-stability; Dahlquist's Theorem 177

b) Stiff stability. 178

c) Gear's methods 181

d) The Liniger-Willoughby methods 186

Summary Remarks. 198

Bibliography 202

Appendix I: Lecture Timetable 205

Appendix II: Errata in Text 206

NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS: LECTURE NOTES

Part I: Survey

1. The problem; existence

The object of interest here is the first-order ordinary differential equation (ODE):

$$\dot{y} = f(y,t)$$

Here t is the independent variable, which may or may not be time, and $\dot{y}$ (or any symbol with a dot over it) means the derivative with respect to t. The quantities y, f, and $\dot{y}$ are to be vectors of length N. The case N=1 is simply that of a scalar ODE. Since most of the subject generalizes immediately to the case of ODE systems (N>1), and virtually all applications of the subject are to systems, we will assume a general N here, unless stated otherwise.

In many applications the ODE's of interest are of higher order than the first, e.g.

$$\ddot{u} = g(u,\dot{u},t).$$

The most common practice then is to reduce such an equation to a first-order system:

$$y = \begin{pmatrix} y^1 \\ y^2 \end{pmatrix} = \begin{pmatrix} u \\ \dot{u} \end{pmatrix}, \quad \dot{y} = \begin{pmatrix} \dot{u} \\ \ddot{u} \end{pmatrix} = \begin{pmatrix} y^2 \\ g(y^1, y^2, t) \end{pmatrix} = f(y,t)$$

and apply the methods for first-order ODE's. Occasionally, special methods for higher-order equations are used instead.

In any case, the right-hand side function $f(y,t)$ must be a known vector function of the vector y and the scalar t. Problems in which $\dot{y}(t)$ involves past values of $y(\tau)$, by way of in an integral or some other mathematical relation, do not fall under the present subject, but instead are integro-differential equations, functional-differential equations, or something else. In the case of a system, the function $f(y,t)$ includes the coupling of the various components, in that the i^{th} component of $\dot{y}$, namely $\dot{y}^i$, will in general depend on other y^j as well as on y^i.

The general solution an ODE system of size N (like that of an N^{th} order scalar ODE) will in general have N free parameters in it, which must be specified on the basis of other information about the problem. Here we will assume that that information consists of $y(t_0) = y_0$, the vector of values of y at t_0, the initial value of t. Hence, we are solving an initial value problem on an interval $t_0 \leq t \leq T$, in which $y(t)$ is to be determined from y_0 and the ODE. The final value T may be ∞, or it may not be known at the start of the problem.

Typical of the areas in which these problems arise are: chemistry, where the ODE may represent the kinetis of a system of species undergoing chemical reactions; electronics, where it may represent the response (in time) of a circuit; particle physics, where it may represent the equations of motion of particles under various forces; astronomy, where it may represent the time evolution of a star, galaxy, or the whole universe; and partial differential equations in space and time, where the spatial derivatives have been treated by finite differencing or other techniques, resulting in an ODE system in time.

It will seldom be necessary to draw from the theory of ODE's here, as opposed to numerical methods. However, there is one theorem from ODE theory that provides the rigorous mathematical foundation for all that follows. It is the fundamental existence theorem, and it is roughly stated as follows:

Theorem: If $f(y,t)$ satisfies a Lipschitz condition in y and is continuous in t for $t_0 \leq t \leq T$, then the initial value problem $\dot{y}=f(y,t)$, $y(t_0)=y_0$ has a unique solution.

The meaning of the Lipschitz condition is that for any two vectors y and $\bar{y}$, we have

$$|f(y,t) - f(\bar{y},t)| \leq L\,|y-\bar{y}|,$$

where L is a constant, the Lipschitz constant. If the problem is a scalar one, then the quantities above are just absolute values. In the vector case, they are norms, which generalize the concept of absolute value to vectors. Many kinds of norms are used: e.g. the Euclidean norm $\left(|v| = \sqrt{\sum_1^N (v^i)^2}\right)$, or the maximum norm $\left(|v| = \max |v^i|\right)$. Any such norm is acceptable for the present purposes.

The Lipschitz condition simply quarantees that f is not too wildly behaved as a function of y. In particular it is continuous in y. The necessity for such a condition in the existence and uniqueness theorem above can be seen from the following simple example. The scalar problem

$$\dot{y} = y^{2/3},\ y(0) = 0$$

has the trivial solution $y = 0$, but also the solution $y = (t/3)^3$. Uniqueness of the solution, as guaranteed in the theorem, does not hold

here, since $y^{2/3}$ does not satisfy a Lipschitz condition.

In applications, a problem in which a Lipschitz condition is absent will also cause numerical methods to fail. The fault is not with the methods but with the ill-posed nature of the problem itself. A problem that is well-posed on physical grounds should lead to one that is also well-posed mathematically. If it does not, the mathematical model being used is probably a poor one.

2. Discretization

Numerical methods for ODE's tend to fall into two broad categories: discrete and non-discrete. These descriptions refer to the set of points at which a numerical solution is produced by the method. The discrete methods produce approximate values of $y(t)$ at discrete values of t, i.e. at a set of mesh points on the t axis. These points may or may not be equally spaced, but always form a finite sequence. The non-discrete methods produce approximations to $y(t)$ on entire intervals of the t axis. That is they produce entire curves, or pieces of curves which are fitted together to form the approximate solution.

The distinction between discrete and non-discrete methods becomes somewhat blurred in their implementations. Discrete methods are often accompanied by interpolation formulas that allows one to calculate the approximate $y(t)$ at arbitrary non-mesh values. At the same time, non-discrete methods usually utilize an underlying mesh on which the solution curve is constructed in pieces. Nevertheless, the basic discreteness or non-discreteness of a given method is usually apparent on close examination.

The class of non-discrete methods is the smaller and less commonly used of the two. It includes the Taylor series methods, the spline methods, and the Lie series methods. Some mention of these will be made later (Chapter 5). The discrete methods include the Runge-Kutta and linear multistep methods, and it is the discrete class that we will pay virtually all of our attention to here.

3. Basic concepts; explicit Euler method

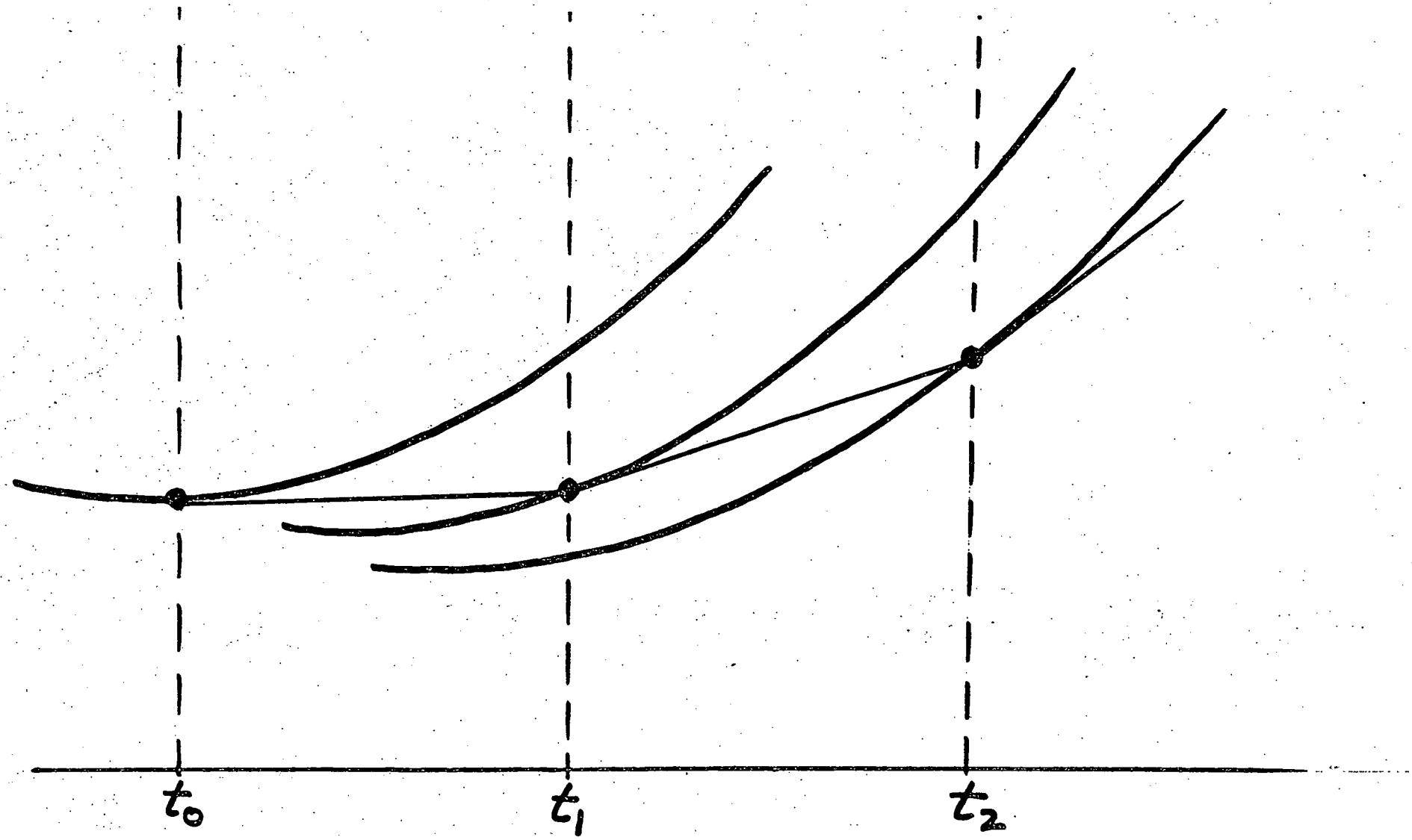
The method of Euler is probably the simplest and oldest (dating to the 1700's) of all numerical methods for ODE's. Yet despite this, and the fact that it is rarely used today, it serves to illustrate many of the basic concepts that are important for all methods.

The Euler method, or the explicit Euler method (to distinguish it from certain variants of it), consists of dividing the interval $t_0 \le t \le T$ into equal parts of length h, the stepsize, and applying the formula

$$y_{n+1} = y_n + h\dot{y}_n$$

successively for n=0, 1, 2, ---. Here $\dot{y}_n$ stands for $f(y_n,t_n)$, and $t_n=t_0+nh$. We assume f satisfies the continuity and Lipschitz condition necessary for the existence theorem to apply. Then the method produces y_1, y_2, --- from y_0 and the given ODE, y_n being an approximation to $y(t_n)$.

Graphically (see figure) this method consists, at the n^{th} point, of drawing the tangent at t_n to the curve which represents that solution of the ODE that has the value y_n at t_n, then extending that tangent line from t_n to $t_{n+1} = t_n+h$, and taking y_{n+1} to be the value of y on that line at $t=t_{n+1}$. Thus y_1 requires the tangent to the curve of the solution y(t) for which $y(t_0)= y_0$. But y_2 requires the tangent to a curve z(t), which is in general not the same as y(t), that is a solution of the ODE for which $z(t_1)=y_1$.



Explicit Euler Method

This process is continued until $t_n = T$. The totality of line segments produced is a broken line or polygon approximating the desired true solution $y(t)$; the method is alternatively called the polygon method. If the problem is a system, this construction must be done for each of the N components separately.

3.(a) Explicit vs implicit

The most obvious property of the above Euler method is that it is explicit. This means that y_{n+1} is given by a closed-form formula in terms of values of y, $\dot{y}$, and t at the n^{th} or earlier points. By contrast, implicit methods involve a formula for y_{n+1} that is implicit and requires the solution of an equation or system of equations, generally by some iterative process. These will be illustrated later.

3.(b) Truncation error, order, convergence

It is not sufficient to take a reasonable-looking method and apply it on more or less blind faith. One must (or should) give some forethought to the sources of error that the method contains and to the control of those errors by way of choosing the stepsize h.

To begin with, consider the n^{th} step taken with the explicit Euler method, and suppose that at t_n we start from the true solution curve $y(t)$ that solves the given problem. The step produces the vector

$$y_{n+1} = y(t_n) + hf(y(t_n), t_n),$$

and this is in error by the amount

$$d_n = y(t_n) + hf(y(t_n), t_n) - y(t_{n+1}).$$

This is called the local truncation error--"local" because it does not involve errors arising on any other steps, and "truncation" because it is

it is the remainder after truncating the Taylor series for $y(t_{n+1})$, expanded about t_n, to two terms.

The <u>global truncation error</u> is the vector

$$e_n = y_n - y(t_n),$$

which measures the effect of all the local truncation errors arising in steps 0, 1, ---, n-1, on the calculated quantity y_n after n steps. (Here y(t) is a solution of the original problem with $\dot{y}=f(y,t)$ and $y(t_0)=y_0$.) To see how this effect occurs, we write

$$e_{n+1} = y_{n+1} - y(t_{n+1}) = y_n + h\,f(y_n, t_n) - y(t_{n+1})$$

and subtract e_n from this to get

$$e_{n+1} - e_n = h f(y_n, t_n) + y(t_n) - y(t_{n+1})$$

$$e_{n+1} = e_n + d_n + h[f(y_n, t_n) - f(y(t_n), t_n)].$$

At this point the Lipschitz condition, together with the triangle inequality, becomes very useful, since it allows us to write

$$|e_{n+1}| \le |e_n| + |d_n| + hL\,|y_n - y(t_n)|$$

$$= |e_n|(1+hL) + |d_n|,$$

where L is the Lipschitz constant for f. This recursive inequality on the global errors shows that the global error at t_{n+1} is bounded by the local error committed on the n<u>th</u> step plus the global error already present at t_n, magnified by the factor 1+hL.

Return for a moment to the local error d_n itself. By writing the Taylor series

$$y(t_{n+1}) = y(t_n) + h\,\dot{y}(t_n) + \frac{h^2}{2}\,\ddot{y}(t_n) + O(h^3),$$

we conclude that

$$d_n = -\frac{h^2}{2}\,\ddot{y}(t_n) + O(h^3) = O(h^2).$$

That is

$$|d_n| \le D = D(h),$$

where $D(h)/h^2$ is bounded as h varies. More precisely, D/h^2 is bounded by the maximum of $|\ddot{y}(t)|/2$. The h^2 behavior of the local error means that the method is of order 1, the order being one less than the exponent on h.

We need now to accumulate the local errors by the recursive inequality to get a bound on the global error. Letting $K = 1+hL$ we have

$$|e_{n+1}| \le K|e_n| + D$$

$$|e_1| \le K|e_0| + D$$

$$|e_2| \le K(K|e_0|+D) + D = K^2|e_0| + D(K+1)$$

$$|e_3| \le K(K^2|e_0| + D(K+1)) + D$$
$$= K^3|e_0| + D(K^2+K+1)$$

$$\cdots$$

$$|e_n| \le K^n|e_0| + D(K^{n-1} + \cdots + K + 1)$$
$$= (1+hL)^n|e_0| + D\frac{(1+hL)^n - 1}{hL}$$

Using the relation $1+hL \le e^{hL}$, we have

$$(1+hL)^n \le e^{hLn} \le e^{bL},$$

where b is the length of the interval, $b = T - t_0$. Thus

$$|e_n| \le e^{bL}|e_0| + \frac{D}{hL}(e^{bL} - 1).$$

Recalling that D/h is O(h) and so tends to zero as $h \to 0$, we conclude that if $|e_0| \to 0$ and $h \to 0$, all $|e_n|$ tend to 0. That is, the method is convergent in the sense that a small enough stepsize and a small enough initial error $e_0 = y_0 - y(t_0)$ (if any) will lead to an arbitrarily small global truncation error for all calculated points.

It should be noted that b is assumed to be finite here, and is kept fixed. Thus this result must be given a special interpretation in the case of an infinite interval: we have only proved convergence on any finite part of that interval.

On the other hand, the bound on $|e_n|$ is somewhat pessimistic in that it represents the accumulation of many inequalities. It is unlikely that all of the relevant inequalities (including those from the Lipschitz condition) were close to being equalities, and so it is unlikely that $|e_n|$ will be near its bound.

3. (c) Error estimation

The pessimism in the global error bound can be largely eliminated by the use of an auxiliary ODE to estimate e_n. Return to the relations

$$e_{n+1} = e_n + d_n + h\left[f(y_n, t_n) - f(y(t_n), t_n)\right],$$

$$d_n = -\frac{h^2}{2}\ddot{y}(t_n) + O(h^3),$$

and assume that $e_0=0$, or that the exact initial value is used. We may replace the difference between the two f values above by $f_y\,(y(t_n),t_n)(y_n-y(t_n)) + O(h^2)$, where f_y is the matrix of partial derivatives of f with respect to y. (This uses a slightly generalized mean value theorem.) Then we end up with

$$e_{n+1} = e_n - \frac{h^2}{2}\ddot{y}(t_n) + h f_y e_n + O(h^3)$$

If we now set $\delta_n = e_n/h$, this becomes

$$\delta_{n+1} = \delta_n + h\left[f_y \delta_n - \tfrac{1}{2}\ddot{y}(t_n)\right] + O(h^2).$$

Except for the final error term, this can be recognized as the application of Euler's method to the ODE

$$\dot{\delta}(t) = f_y(t)\delta(t) - \tfrac{1}{2}\ddot{y}(t)$$

In fact, the same convergence proof as used earlier can be applied here to show that as $h \to 0$, δ_n converges to $\delta(t_n)$ where $\delta(t)$ is the solution to the above ODE with the initial value $\delta(t_0)=0$. If $y \in C^3$ (y has 3 continuous derivatives) then one can show the slightly stronger result that

$$\delta_n - \delta(t_n) = O(h)$$

or that

$$e_n = h\,\delta(t_n) + O(h^2)$$

as $h \to 0$. An example in the text* shows that the estimate $h\,\delta(t_n)$ can be a much more accurate estimate of the global error than the bound derived earlier.

The auxiliary ODE can be used as a means of estimating the global errors in practice. However, it requires a knowledge of f_y and $\ddot{y}$, which were not needed to solve the original problem, and may be expensive to compute. We do have, from $\dot{y} = f$, the relation

$$\ddot{y} = f_t + f_y \dot{y} \qquad \left(f_t \equiv \frac{\partial f}{\partial t}\right),$$

so that now f_y and f_t are required. If it happens that these partial derivatives are easy to come by, and one is willing to integrate a second ODE system of size N as well as the original problem, then this approach yields a more accurate estimate of e_n than any other, when the exact solution is not known in advance.

For the vast majority of realistic ODE problems, it is impractical to attempt to estimate global errors accurately, and one must settle for estimates of local error only. This is nearly always a relatively

*References to the text in these notes are references to C.W. Gear's book, Ref. 1.

inexpensive byproduct of the calculation of the solution itself, and is especially so for the Euler method. Recall that

$$d_n \approx -\frac{1}{2} h^2 \ddot{y}(t_n) \quad \text{(approximate equality)}.$$

But since $\ddot{y}$ is not generally available, we use a numerical approximation to this derivative by using a mean value theorem in the form

$$\dot{y}(t_n) - \dot{y}(t_{n-1}) \approx h \ddot{y}(t_n).$$

The $\dot{y}$ terms above are approximated by the $\dot{y}_n$ and $\dot{y}_{n-1}$ that arise in the method, and so we may write

$$d_n \approx -\frac{1}{2} h (\dot{y}_n - \dot{y}_{n-1}).$$

This is in fact an estimate of the local truncation error that is accurate to within $O(h^3)$. Moreover, it requires only that $\dot{y}$ be saved from the previous step; almost no extra calculation is necessary.

Beyond simply estimating the local error, there are several important things one can (and should) do. One is to test that estimate against some parameter that specifies how much local error is to be tolerated. If the test fails, h should be reduced to a point where it will pass. Furthermore, if the test passes the first time, one should consider increasing h to a value which, while not causing the test to fail, will still improve the efficiency of the calculation by causing larger steps to be taken. In the case of either a decrease or increase in h, the fact that d_n behaves like h^2 (at least asymptotically, in the limit of small h) gives a means of choosing the new value of h, according to the results of the test on the error estimate. More will be said on error control later.

3. (d) Roundoff Error

In the explicit Euler method, the global error obeys

$$|e_n| \le K_1 |e_0| + K_2 h,$$

$$K_1 = e^{bL}, \quad K_2 = \frac{e^{bL}-1}{L}\frac{D}{h^2}, \quad \frac{D}{h^2} = \max|\ddot{y}/2|,$$

if machine roundoff is ignored. However, because the machine can only represent real numbers to finite precision, there is a contribution to the local error of

$$r_n = \epsilon_n y_n u, \qquad |\epsilon_n| \le \frac{1}{2}$$

u = unit roundoff

$= 2^{-48} \approx 10^{-14}$ on CDC machines,

arising from the representation of y_n as $y_n \pm r_n$ in the machine. If A = max $|y|$, then $|r_n| \le \frac{1}{2} Au$, and the effect on $|e_n|$ is to replace D (the bound on $|d_n|$) by $D + \frac{1}{2} Au$. The resulting bound is

$$|e_n| \le K_1|e_0| + K_2 h + K_3 u/h,$$

$$K_3 = \frac{e^{bL}-1}{2L} A.$$

This bound does <u>not</u> vanish as h gets smaller, and instead becomes infinite. This reflects the fact that if an error of a constant magnitude is introduced at each of the n steps, and n becomes infinite, then the total error does also. However, in most applications, the range of values of h actually used is such that the roundoff term above is negligible compared to the truncation term.

To illustrate this, consider the case b = 1 and $K_2 = K_3$. Our bound on $|e_n|$, as a function of h, has a minimum at $h = \sqrt{u} \approx 10^{-7}$. It is unlikely that as many as 10^{-7} steps will be taken, and so it is unlikely that the roundoff term is significant.

The worst-case analysis above is quite pessimistic in that the r_n are of somewhat random signs, so that their cumulative effect is approximately the sum of n random numbers r_n, distributed in the interval $|r_n| \leq \frac{1}{2}$ Au. If the distribution is uniform, the expected magnitude of this sum is about

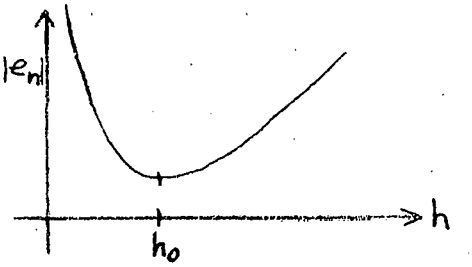
$$\tfrac{1}{2} A u \sqrt{n/3}$$

This makes the roundoff term proportional to $1/\sqrt{h}$. For the example above, the minimum of the bound is then at

$$h = (u/2)^{2/3} \approx \tfrac{1}{7} 10^{-9}.$$

It is highly unlikely that we will want to take 7 billion steps to complete the problem, and it becomes all the more valid to ignore roundoff.

Assuming a simple form for the contribution of roundoff error, such as one of the two above, we can plot the final total error as a function of h. To the right of the minimum point h_0, the curve would approximate that given by the truncation error terms above, and to the left of h_0, it would climb to infinity as $h \to 0$, because of the roundoff term. The justification for ignoring roundoff is that, say on a machine with 14 decimal places, the region of actual values of h used is usually far to the right of h_0.



3. (e) Stability

The subject of stability is in general the study of the effects of perturbations in the values at one point on the calculated values later. If the effects are small, we regard the calculation process as a stable one. There are several well-defined properties in particular that we look for as indicators of stability in ODE methods. In illustrating these, we will ignore the contribution of roundoff.

The reason for studying the numerical effects of perturbations is not that we anticipate making such perturbations consciously, but that they will inevitably be made during the numerical solution anyway. Each source of error -- truncation, roundoff, etc., that occurs on every step, acts as a perturbation on the values used in performing the subsequent steps. Thus if a total error of e_n has been generated in n steps (even though there was none initially), we can regard e_n as a perturbation in y_n and study its effects on the calculated values y_m for any $m > n$. In this way, the subject of stability is very much related to that of errors and their growth. Although we attempt to study it separately

Consider solving the ODE $y' = f(y,t)$ with the explicit Euler method, and consider a perturbation e_0 in the initial value y_0. Thus we start at $y_0 + e_0$ and use the method to generate a sequence of calculated values $z_1, z_2, z_3, \cdots, z_n, \cdots$. If we write $z_n = y_n + e_n$ and subtract the formulas

$$y_{n+1} = y_n + h f(y_n, t_n)$$
$$z_{n+1} = z_n + h f(z_n, t_n)$$

we get

$$e_{n+1} = e_n + h\left[f(y_n+e_n, t_n) - f(y_n, t_n)\right]$$

$$|e_{n+1}| \le |e_n| + hL|e_n|,$$

using the Lipschitz condition. We may accumulate these inequalities as before to get

$$|e_n| \leq (1+hL)^n |e_0| \leq e^{bL} |e_0|.$$

This says that the change e_n in any calculated value (in the finite t interval of length b) is bounded by a multiple of the norm of the initial perturbation e_0. When this happens for a given numerical method, we say simply that the method is stable. In this case, the stability result is independent of h, but more often it will only hold (and will only be required) for all sufficiently small values of h.

This particular property of stability is not of sufficient value when bL can be large, and especially when b can be infinite. In such cases, it would be desirable to have, for example, the property that $|e_{n+1}| \leq |e_n|$ for the perturbations. This property, however, may or may not hold, depending on the problem being solved, as well as on the method used.

In order to restrict the class of problems enough to study this kind of stability, we consider the case of a linear, constant-coefficient, homogeneous ODE: $\dot{y} = Ay$, where A is a constant N x N matrix. We justify this (seemingly great) reduction in generality by performing a local analysis of the general ODE $\dot{y} = f$. If we work in a small neighborhood of a point (y_0, t_0), we can approximate this ODE by the linearized ODE

$$\dot{y} = f(y_0, t_0) + f_y(y_0, t_0)(y - y_0) + f_t(y_0, t_0)(t - t_0).$$

This has the form $\dot{y} = Ay + b(t)$, and the inhomogeneous term $b(t)$ can be transformed away by a change of variables to $z = y - a(t)$ for some $a(t)$.

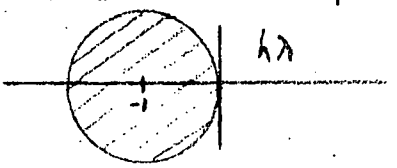
Now from the ODE $\dot{y} = Ay$, we simplify further by diagonalizing the matrix A, i.e. by writing $A = PDP^{-1}$, where P is some other matrix and D is a diagonal matrix with diagonal elements $\lambda_1, \text{---}, \lambda_n$, which are the eigenvalues of A. This diagonalization can be performed for large classes of matrices A (for example when the eigenvalues of A are all distinct), and we will assume it can be done here. Then the change of variables to $z = P^{-1}y$ leads to $\dot{z} = Dz$, which is an uncoupled set of scalar ODE's, each of the form $\dot{y} = \lambda y$. Here λ is any of the eigenvalues, and as such can be a complex number.

What is often done, therefore, in the study of stability properties of ODE solution methods, is to consider only this simple "test equation" with an arbitrary complex constant λ. The above digressions into ODE transformations and linear algebra are needed only to motivate the study of this ODE, not to carry out that study.

For $\dot{y} = \lambda y$, the results of the explicit Euler method are trivial to write down: $y_{n+1} = y_n + h\lambda y_n = (1+h\lambda)\, y_n$. Moreover, the equation for the perturbations is the same: $e_{n+1} = (1+h\lambda)\, e_n$. Now we can easily say that the property $|e_{n+1}| \leq |e_n|$ will hold if and only if h and λ are such that $|1+h\lambda| \leq 1$. Note that this property depends only on the product $h\lambda$, and this will be true for most methods generally. We can draw the complex $h\lambda$ plane and in it the region

$$S_a = \{h\lambda : |1 + h\lambda| \leq 1\},$$

which is called the absolute stability region for this method. It is simply the disk of radius 1 about the point -1.



When $h\lambda \in S_a$, the method possesses absolute stability for the test equation, in the sense that the perturbations decrease in magnitude from step to step. (The term "absolute" is used to distinguish from relative stability, in which perturbations are studied relative to the solution itself or relative to some approximation to the solution. The latter does not arise until we discuss multistep methods.) The absolute stability property is of interest only when $Re(\lambda)<0$, i.e. when $y(t)$ is decaying.

As shown in the text (p.17), the problem $\dot{y}=2t-1000(y-t^2)$, $y_0=0$, where $\lambda = -1000$, and inhomogeneous terms are still present, can not be solved accurately by the explicit Euler method when $h\lambda \notin S_a$, i.e. when $h > .002$. This illustrates the usefulness of the absolute stability region, and also the fact that the presence absolute stability is often correlated with the absence of large errors.

For the problem $\dot{y} = \lambda y$, $y(o) = y_o$, we can study the growth of errors more deeply, because we know both the true solution, $y(t) = e^{\lambda t} y_0$, and the calculated solution, $y_n = (1+h\lambda)^n y_0$. Since $t_n = nh$, we have

$$y_n = \left[(1+h\lambda)^{1/h\lambda}\right]^{\lambda t_n} y_o = a^{\lambda t_n} y_o ,$$

$$y(t_n) = e^{\lambda t_n} y_o .$$

Thus the global error in y_n is measured by the accuracy with which $a = (1+h\lambda)^{1/h\lambda}$ approximates the constant e. It is well known that $a \to e$ as $h \to o$, verifying again that the method is convergent here.

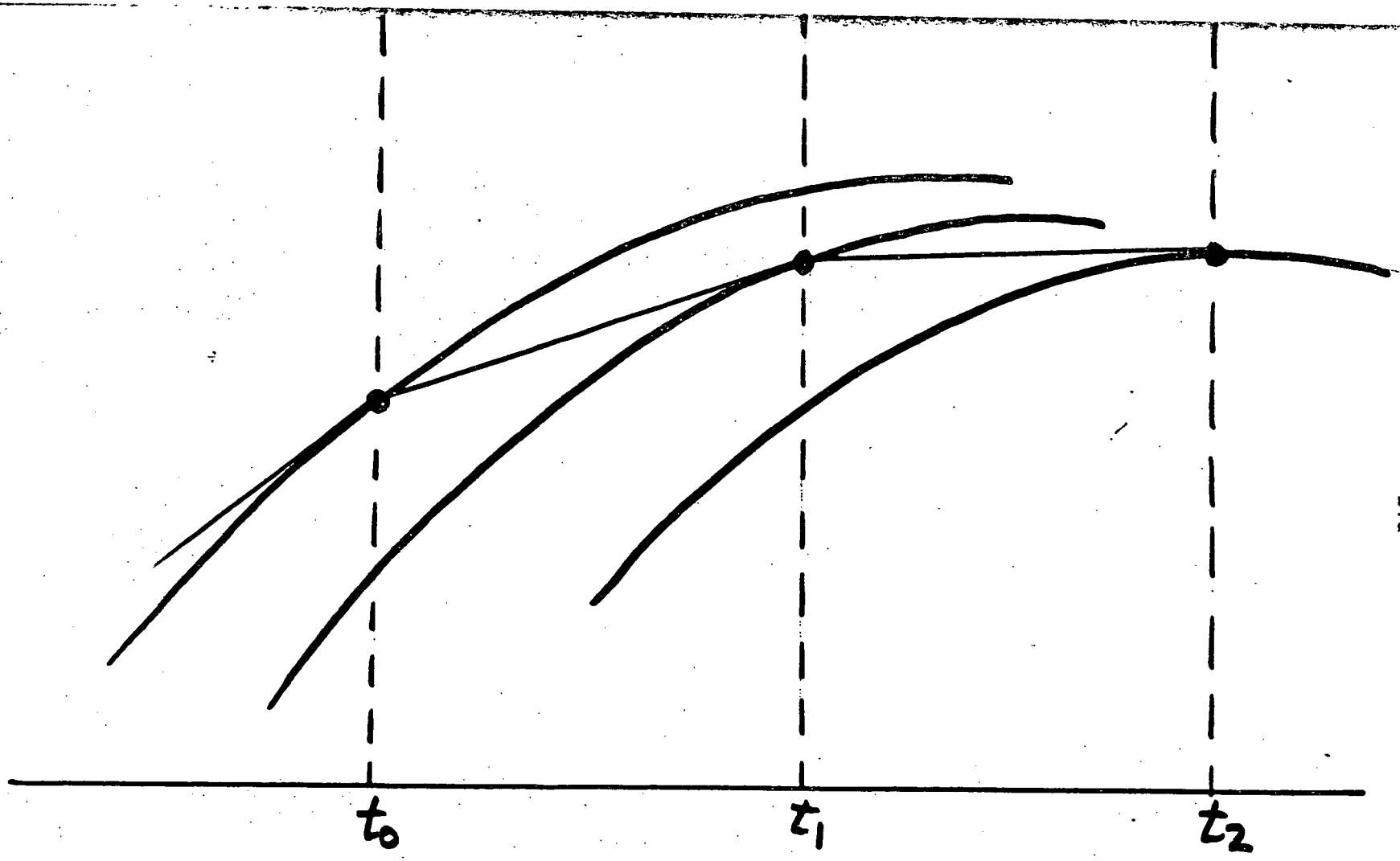
3. (f) Efficiency

The efficiency of the explicit Euler method is rather transparent. The primary cost of taking a step is that of one evaluation of the function f. The remainder of the cost, which is the overhead, consists of a multiplication (of f by h) and an addition (to y_n). Since every step entails the same cost, the total cost is simply $\frac{b}{h}$ (=the number of steps) times the cost per step.

In studying other methods, will we usually consider the total number of evaluations of f as the primary measure of efficiency. This will allow us to compare different methods, assuming that the overhead costs are either negligible or are at least nearly equal for the methods being compared.

4. The implicit Euler method

The simplest of the implicit methods is the backward form of the Euler method, namely the implicit Euler method, given by $y_{n+1} = y_n + h\dot{y}_{n+1}$. The graphical representation here (see figure) is gotten from that of the explicit Euler method by turning it upside down (or backwards). Thus at each step, the line drawn from one calculated point to the next is tangent to a particular solution curve at the forward point.



Implicit Euler Method

As $\dot{y}_{n+1} = f(y_{n+1}, t_{n+1})$, the equation for y_{n+1} must be solved by some means. One way of doing this is to approximate y_{n+1}, say by the explicit Euler formula, evaluate f at that approximation, giving a second approximation to y_{n+1} from the above formula, and repeat this process. Another way is to apply the generalization of Newton's method to the problem of finding the (vector) root of a vector-valued function of N variables. More will be said of this problem later.

In any case, since f is generally quite nonlinear, the calculation of y_{n+1} will require some iterative process. Hence there is another source of error introduced at every step: the iteration error. This is the difference between the true solution y_{n+1} of the implicit formula and the result of a finite number of iterations in approximating that solution.

Iteration error can be viewed in either of two ways. First, if only one or two iterations are to be performed, then the iteration formulas themselves can be combined with the formula for the initial guess, giving an explicit method, which can be analyzed as such. An example would be the pair of formulas

$$\tilde{y}_{n+1} = y_n + h\dot{y}_n$$

$$y_{n+1} = y_n + h f(\tilde{y}_{n+1}, t_{n+1})$$

which can be combined to

$$y_{n+1} = y_n + h f(y_n + h\dot{y}_n, t_{n+1}).$$

On the other hand, if we are willing to perform iterations until the iteration error is much smaller that the local truncation error associated with the method, then all properties of the original implicit method are

valid, and the iteration error can be ignored for all practical purposes. This latter view will be taken here.

The local truncation error for the implicit Euler method is defined to be

$$d_n = y(t_n) + hf(y(t_{n+1}), t_{n+1}) - y(t_{n+1}).$$

By expanding this in a Taylor series about t_{n+1}, we get (much as before)

$$d_n = \frac{h^2}{2}\ddot{y}(t_n) + O(h^3) = O(h^2).$$

Hence, the <u>order</u> of the method is again 1.

The actual error committed in a single step is not d_n, as it was for the explicit method. Instead it is the difference

$$\epsilon_n = y_{n+1} - y(t_{n+1})$$

obtained when $y_n = y(t_n)$ is exact, and this differs slightly from d_n. From the relations

$$d_n = y_n + hf(y_{n+1} - \epsilon_n, t_{n+1}) - y(t_{n+1}),$$
$$0 = y_n + hf(y_{n+1}, t_{n+1}) - y_{n+1},$$

we have

$$d_n = h[f(y_{n+1} - \epsilon_n, t_{n+1}) - f(y_{n+1}, t_{n+1})] + \epsilon_n.$$

This equation could in principle be solved for ϵ_n, although this cannot be done in closed form because of the implicit occurrence of ϵ_n in the argument of the first f. If that were done, the result would be that ϵ_n is equal to d_n within higher order terms (which are $O(h^3)$), and so in particular $\epsilon_n = O(h^2)$. Using this fact and expanding a Taylor series above, we can write

$$d_n = -h f_y(y_{n+1}, t_{n+1})\epsilon_n + \epsilon_n + O(h^5)$$

$$\epsilon_n = [I - h f_y(y_{n+1}, t_{n+1})]^{-1} d_n + O(h^5) = d_n + O(h^3).$$

Here I is the N x N identity matrix. This shows more explicitly the agreement between ϵ_n and d_n. Thus to the order of the leading term in these quantities, d_n is an equally good approximation to the error in y_{n+1} corresponding to local truncation error. The reason for preferring d_n to ϵ_n is that d_n is defined explicitly for a given solution y(t), while ϵ_n is not. Hence d_n is the easier quantity to work with in practice.

The global errors $e_n = y_n - y(t_n)$ can be analyzed much as before. Here y(t) represents the fixed true solution of the original initial value problem. By writing the various defining equations for e_n, e_{n+1}, and d_n, we obtain

$$|e_{n+1}| \leq (|e_n| + D)/(1 - hL),$$

provided $|d_n| \leq D$ and $hL < 1$. This recursive inequality has the same form as before, and we can combine these to get

$$|e_n| \leq \frac{|e_0|}{(1-hL)^n} + \frac{D}{1-hL}\,\frac{(1-hL)^{-n} - 1}{(1-hL)^{-1} - 1}$$

$$= (1-hL)^{-n}|e_0| + \frac{D}{hL}\left[(1-hL)^{-n} - 1\right].$$

Now use the fact that $(1 - hL)^{-1} \leq e^{2hL}$ when $h \leq h_0$ for some constant h_0. Hence

$$|e_n| \leq e^{2nhL}|e_0| + \frac{D}{hL}\left(e^{2nhL} - 1\right).$$

If $|\ddot{y}| \le C$, then we can use $D = Ch^2/2$, and the fact that $nh \le b$, to get

$$|e_n| \le e^{2bL}|e_0| + h\frac{C}{2L}(e^{2bL}-1) \quad (h \le h_0).$$

This proves that the method is convergent.

The restriction $hL < 1$ is essential for this error bound. For example, the method fails for the problem $\dot{y} = \lambda y$ with $h\lambda = 1$, because the formula reduces to an absurdity. Likewise, if $h\lambda > 1$, the method produces oscillating answers that become increasingly inaccurate. However, if $\lambda < 0$ the method works well, despite the fact that $hL = h|\lambda|$ may be larger than 1.

To show that the implicit Euler method is stable, we consider a perturbation e_0 in y_0, and look at the resulting perturbations e_n in y_n. We get

$$y_{n+1} + e_{n+1} = y_n + e_n + hf(y_{n+1} + e_{n+1}, t_{n+1})$$

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1})$$

$$e_{n+1} = e_n + h[f(y_{n+1} + e_{n+1}, t_{n+1}) - f(y_{n+1}, t_{n+1})]$$

$$|e_{n+1}| \le |e_n| + hL|e_{n+1}|$$

$$|e_{n+1}| \le \frac{|e_n|}{1-hL}$$

$$|e_n| \le (1-hL)^{-n}|e_0| \le e^{2bL}|e_0|$$

for h sufficiently small.

The absolute stability region is given by looking at the equation $\dot{y} = \lambda y$, for which the perturbations satisfy

$$e_{n+1} = \frac{e_n}{1-h\lambda}$$

Hence

$$S_a = \{h\lambda : \left|\frac{1}{1-h\lambda}\right| \leq 1\} = \{h\lambda : |1-h\lambda| \geq 1\}.$$

Thus we have much better stability properties here than in the explicit case.

If, for arbitrary h and λ with $Re(\lambda) < 0$, a method applied to $\dot{y} = \lambda y$ gives answers that satisfy $y_n \to 0$ as $n \to \infty$, the method is called A-stable. Thus we see that the implicit Euler method is A-stable, while the explicit one is not.

The question of efficiency is more complicated here than in the explicit case. Generally, in solving for y_{n+1} by an iterative method, one evaluation of f is required at each iteration, and so the number of f evaluations per step is just the number of iterations per step. The overhead, which includes all other costs, is considerably higher than with the explicit method, because of the iteration. For example, if Newton's method is used, the overhead includes the solution of an N x N linear system each iteration.

Exercise 4.1

Prove that the global errors for the implicit Euler method satisfy

$$|e_{n+1}| \leq \frac{|e_n| + D}{1 - hL}$$

if $|d_n| \leq D$ and $hL < 1$.

Exercise 4.2

Use the implicit Euler method, first with h=1, then with h=1/2, to solve

$$\dot{y} = 2t - 1000\,(y - t^2), \quad y(0) = 0, \quad 0 \leq t \leq 1.$$

What are the errors at t = 1?

5. Survey of method classes

This brief description of the general classes of ODE methods is intended to give a frame of reference from which more detailed discussions can follow later.

5. (a) Discrete vs nondiscrete

As mentioned earlier, discrete methods are those whose end result is a sequence $y_0, y_1, \ldots, y_n, \ldots$, where y_n approximates $y(t_n)$ and $\{t_n\}$ is a mesh of values of t. By contrast, nondiscrete methods have an end result which is a complete curve or system of curves representing the approximate solution. The boundary between these classes is blurred somewhat in the implementation of the various methods. Nondiscrete methods generally use a mesh, and they often use data at discrete points to determine the curves. Discrete methods often involve interpolating functions for the evaluation of approximate values of y(t) for arbitrary t.

5. (b) One-step vs multistep

A second dichotomy, within the class of discrete methods, is made between one-step and multistep methods. In general, if calculated values up to and including the nth mesh point have been obtained, then the next one is given by

$$y_{n+1} = \varphi(t_{n+1}, y_n, t_n, y_{n-1}, t_{n-1}, \cdots, y_{n+1-k}, t_{n+1-k})$$

where φ is some function involving f, possibly only implicitly defined. Here k is the number of steps involved, and the method is called a k-step method.

When $k > 1$, we have a multistep method, and it is necessary to start up the calculation by some special technique, because the required past history is not initially available. One-step methods do not suffer from this difficulty.

5. (c) Runge-Kutta

Within the class of discrete one-step methods, those of Runge-Kutta type are characterized by the fact that they involve evaluations of f at intermediate values of t, rather than just at the mesh points t_n. These intermediate values are auxiliary only, not intended for use as approximate solution values. The classical Runge-Kutta method is given by the formulas

$$k_0 = hf(y_n, t_n)$$

$$k_1 = hf(y_n + \tfrac{1}{2} k_0, t_n + \tfrac{1}{2} h)$$

$$k_2 = hf(y_n + \tfrac{1}{2} k_1, t_n + \tfrac{1}{2} h)$$

$$k_3 = hf(y_n + k_2, t_n + h)$$

$$y_{n+1} = y_n + (k_0 + 2k_1 + 2k_2 + k_3)/6$$

Note the intermediate t values in k_1 and k_2.

Runge-Kutta methods can be either explicit or implicit. Explicit ones, typified by the above, are those where $y_{n+1} = \varphi$ can be written in closed form. Implicit ones require the solution of an algebraic equation or system of equations in the calculation of y_{n+1}, as would occur, for example if the formula for k_2 above were replaced by

$$k_2 = hf(y_n + \tfrac{1}{4} k_1 + \tfrac{1}{4} k_2, t_n + \tfrac{1}{2} h).$$

5. (d) Linear multistep

This class of discrete methods is also referred to as the finite difference methods. These are based on formulas of the form

$$y_n = \sum_{i=1}^{K_1} \alpha_i \, y_{n-i} + h \sum_{i=0}^{K_2} \beta_i \, \dot{y}_{n-i}$$

where $\dot{y}_k = f(y_k, t_k)$. If $K = \max(K_1, K_2)$, then this formula represents a K-step method. It is explicit if $\beta_0 = 0$, and implicit otherwise.

The coefficients α_i and β_i are usually constants associated with the method. They are chosen so as to attain desired properties such as order and stability. Linear multipoint formulas can often be obtained from numerical quadrature formulas, in which $\dot{y}(t)$ is a known integrand and $y(t_n)$ is to be approximated by a linear combination of values of $\dot{y}$ and known previous values of y.

If h is assumed constant, then the α_i and β_i are constant. But if the past mesh points t_{n-i} are not equally spaced, and this spacing is taken into account in the method, then the α_i and β_i are functions of the step sizes $h_j = t_{j+1} - t_j$. The formula in that is called a variable-step formula.

In recent years the class of linear multistep methods has been expanded in several ways. One way is the development of composite linear multistep methods, where several different linear multistep methods are used on successive steps, in a cyclic fashion. Another is the generalization to multivalue methods, which result from reformulating the multistep methods in terms of vectors containing the past values and matrices that transform these vectors. A third development is the use of intermediate values of t, as in the Runge-Kutta methods, but in the multistep context. These are called hybrid methods, multipoint Runge-Kutta methods, or modified multistep methods. Of these various developments, only the multivalue methods will be discussed further here (Chapter 9).

5. (e) Extrapolation

The extrapolation methods are based on the idea that approximations based on various step sizes h can be used to extrapolate to the limit h = 0, and thereby gain accuracy without too much extra effort. Specifically, suppose we start with a discrete method with step size h to integrate to the problem from t = 0 to t = T (a multiple of h) and get an approximation to y(T) denoted by y(T,h). We proceed on the assumption that y(T,h) has a power series in h:

$$y(T,h) = y(T) + \sum_{1}^{\infty} \tau_i h^i.$$

If we compute y(T,h) for several h_i (each an integer submultiple of T), we can use these to estimate the first few τ_i, including $\tau_0 = y(T)$, if we ignore the remaining terms in the power series. This would be polynomial extrapolation. Alternatively we could approximate y(T,h) by a rational function of h (the quotent of two polynomials) and get a rational extrapolation method. Here we would evaluate $y(T,h_i)$ as many times as there are unknown coefficients in the rational function and use these to get the value of that function at h = 0.

This process yields an approximation to y(T) given y(0). We can then repeat the process on the interval [T,2T], and continue until the problem is completed.

5. (f) Splines

The use of spline functions in ODE's is a relatively new contribution to the class of nondiscrete methods. The basic idea here is that on a typical subinterval $[t_n,t_n+h]$, a polynomial p(t) is fitted to y(t), and on successive intervals the p's are spliced together at the mesh points. The

fitting is done by making p satisfy m + 1 conditions, where m is the degree of p, and hence p has m + 1 unknown coefficients. Conditions that are used, and names associated with the resulting methods, are as follows:

$$\left.\begin{aligned} p(t_n) &= y(t_n) \\ \dot{p}(\tau_i) &= f(p(\tau_i),\tau_i) \end{aligned}\right\} \quad \text{— collocation}$$

$$\left.\begin{aligned} \ddot{p}(\tau_i) &= f_y f + f_t\big|_{(p(\tau_i),\tau_i)} \\ \dddot{p}(\tau_i) &= \ldots \\ &\text{etc.} \end{aligned}\right\} \quad \text{Hermite}$$

$$\int_{t_n}^{t_n+h} [\dot{p}(t) - f(p(t),t)]\phi_i(t)\,dt = 0 \qquad \text{— Galerkin}$$

In the collocation and Hermite methods, pointwise conditions on p are formulated which are automatically satisfied by the true solution y(t), and the points τ_i used are chosen in the interval $[t_n,t_n+h]$. In the Galerkin methods, the expression $\dot{p} - f(p,t)$, which vanishes identically when p is replaced by y, is made to be orthogonal (in the sense of the integral inner product) to certain basis functions $\phi_i(t)$.

5. (g) Series

Also in the nondiscrete category are the series methods, which compute the solution curve in pieces each of which is given by a truncated series of some type. The simplest such method is the Taylor series method, where one writes

$$y_{n+1} = y_n + h\dot{y}_n + \frac{h^2}{2}\ddot{y}_n + \ldots + \frac{h^r}{r!}y_n^{(r)}$$

$$\dot{y}_n = f(y_n, t_n)$$

$$\ddot{y}_n = f_y f + f_t\Big|_{(y_n, t_n)}$$

etc.

The above would represent an explicit method of order r. The Lie series methods generalize this idea to the situation where y(t) is regarded as a perturbation of a known function $\bar{y}(t)$ and a series for the difference $y - \bar{y}$ is developed. This approach seems to be used only in the case $\dot{y} = f(y)$, where the ODE has no explicit t dependence.

The series methods are generally regarded as being of practical value only when f(y,t) is of a relatively simple form. The reason for this is the rather complex expressions for the derivatives of y that occur. Moreover, their complexity grows rapidly as the order increases and as the complexity of f increases. The increasing use of symbolic manipulation on computers, and symbolic differentiation in particular, may increase the range of practicality of series methods, and for problems in that range, these methods seem to be competitive with others. However, it seems quite unlikely that the general problem will ever be solvable by series methods.

5. (h) List of methods

The following is a brief structured list of methods for the numerical solution of first-order ODE's (including ODE systems). Included in parentheses are names associated with prominent examples. Neither the list nor the examples named are intended to be exhaustive. The list is intended only for purposes of quick reference.

I. Discrete Methods

A. Runge-Kutta Methods

1. Explicit (classical, Gill, Heun, Ralston, Lomax, England, Fehlberg, Luther, Merson)
2. Semi-implicit (Rosenbrock, Calahan)
3. Implicit (Butcher, Ceschino-Kuntzman)

B. Linear Multistep and Multivalue Methods

1. Explicit (Euler, midpoint, Adams-Bashforth)
2. Implicit (Euler, trapezoid, Adams-Moulton, Milne, Hamming, Gear, Liniger-Willoughby)
3. Composite (Bickart-Burgess-Sloat)

C. Hybrid Methods

1. Multistep Runge-Kutta (Byrne-Lambert)
2. Modified multistep (Butcher)

D. Extrapolation Methods

1. Polynomial extrapolation (Neville)
2. Rational extrapolation (Bulirsch-Stoer)

E. Others (Treanor, Lawson)

II. Non-Discrete Methods

A. Series methods

1. Taylor series
2. Lie series (Knapp-Wanner)

B. Spline function methods (Loscalzo, Hulme)

6. Analysis of one-step methods

6. (a) Stability, consistency, convergence, and order

The general form of a one-step method for $y = f(y,t)$ can be written

$$y_{n+1} = y_n + h\psi(y_n, t_n, h).$$

The method is given entirely by the function ψ. In computing y_{n+1}, it involves f, the current quantities y_n and t_n, but not any of the quantities existing prior to t_n. The function ψ is given explicitly if the method is an explicit one. E.g. $\psi(y_n,t_n,h) = f(y_n,t_n)$ for the explicit Euler method. But if the method is implicit, ψ will be defined by an implicit equation, and its existence is only theoretical. The Runge-Kutta methods, whether explicit or implicit, all fall under this category of methods.

Recall the definition of stability given earlier: A method is stable if a perturbation in the initial value causes perturbations in later values that are bounded by a multiple of the initial perturbation, regardless of how small h is. (Of course it is always assumed that f satisfies a Lipschitz condition in such discussions.) For the general one-step method considered here, a statement of stability that directly mimics that given for the explicit Euler method can be given: If $\psi(y,t,h)$ satisfies a Lipschitz condition in y, then the method is stable. For if y_n and z_n are two sequences of calculated values using different initial values, and we write $e_n = z_n - y_n$, then

$$e_{n+1} = [z_n + h\psi(z_n, t_n, h)] - [y_n + h\psi(y_n, t_n, h)]$$

$$= e_n + h[\psi(y_n + e_n, t_n, h) - \psi(y_n, t_n, h)],$$

$$|e_{n+1}| \le |e_n| + hL|e_n|$$

$$|e_n| \le (1+hL)^n |e_0| \le e^{bL}|e_0| \quad (L = \text{Lipschitz constant},)$$

if $nh=t_n=b$ is held fixed. Clearly, regardless of h, $|e_n|$ is a bounded multiple of the initial perturbation $|e_0|$.

Recall also the concept of convergence: The method is convergent if, as we fix $t=nh$ but let $h\to 0$, $n\to\infty$, and $y_0\to y(0)$, the computed values y_n converge to $y(t)$, the true solution value. We can show that this property holds if ψ satisfies a simple and reasonable condition, given in the following definition. We say the method defined by ψ is consistent if $\psi(y,t,o) = f(y,t)$.

Theorem: Let ψ be continuous in y, t, h for $o\le t\le b$, $o\le h\le h_o$ and all y, and Lipschitz in y. Then the one-step method $y_{n+1}=y_n+h\psi(y_n,t_n,h)$ is convergent if and only if it is consistent.

Proof: Let $g(y,t) = \psi(y,t,o)$, and solve the initial value problem

$$\dot{z} = g(z,t), \qquad z(0) = y(0).$$

Because of the Lipschitz condition on ψ (and hence g) this problem has a unique solution. We will show that the calculated values y_n converge to $z(t_n)$ as $h\to 0$, with $t_n=nh$ fixed. Let $e_n=y_n-z(t_n)$, and write

$$z(t_{n+1}) = z(t_n) + h\dot{z}(t_n) + \frac{h^2}{2}c_n, \quad |c_n| \le C = \max|\ddot{z}|.$$

(It is not essential to assume that $z(t)$ have a bounded second derivative, but it makes the proof much easier, and so will be assumed.) Also, assume that ψ is Lipschitz in h, so that

$$|\psi(y,t,h) - \psi(y,t,0)| \le L_1 h.$$

(This assumption is also not essential, but simplifies the proof.)

Then we have

$$e_{n+1} = y_{n+1} - z(t_{n+1})$$

$$= y_n + h\psi(y_n, t_n, h) - \left[z(t_n) + h\,g(z(t_n), t_n) + \frac{h^2}{2}c_n\right]$$

$$= e_n + h\left[\psi(y_n, t_n, h) - \psi(y_n, t_n, 0)\right.$$

$$\left. + \psi(y_n, t_n, 0) - \psi(z(t_n), t_n, 0)\right] - \frac{h^2}{2}c_n$$

$$|e_{n+1}| \le |e_n| + h\left[L_1 h + L|e_n|\right] + \frac{h^2}{2}C$$

$$= (1+hL)|e_n| + h^2 L_2 .$$

Accumulating these inequalities, as done in an earlier calculation, we find

$$|e_n| \le (1+hL)^n |e_0| + h^2 L_2 \frac{(1+hL)^n - 1}{hL}$$

Since $e_0 = 0$, we get, for $nh \le b$,

$$|e_n| \le \frac{hL_2}{L}\left(e^{nhL} - 1\right)$$

$$\le \frac{hL_2}{L}\left(e^{bL} - 1\right) \longrightarrow 0 \quad \text{as} \quad h \to 0.$$

This proves the assertion, $y_n - z(t_n) \to 0$ for fixed nh.

Now if we have consistency, then $g \equiv f$, and so $z(t) \equiv y(t)$, and the above result says that the method is convergent: $y_n - y(t_n) \to 0$.

Conversely, if we have convergence, then for any fixed $t = t_n = nh$, we have $y_n - y(t)$ and $y_n - z(t)$ both $\to 0$ as $h \to 0$, and hence $y(t) = z(t)$.

Since this holds for any t, we have also $\dot{y}(t) = \dot{z}(t)$, which at t=0 says $g(y_0,0) = f(y_0,0)$. But we can carry out this argument starting at any t_0 in (o,b), i.e. starting at any (y_0,t_0) as the initial point, and so get $g(y_0,t_0) = f(y_0,t_0)$. Hence, g≡f, or we have consistency. QED

In order to discuss one-step methods in more meaningful terms, we need to know more about the errors than just convergence.

Definition. The local truncation error of the general one-step method $y_{n+1}=y_n+h\psi(y_n,t_n,h)$ is

$$d_n(h) = y(t_n) + h\psi(y(t_n),t_n,h) - y(t_{n+1}),$$

where y(t) is a true solution of the ODE.

In other words, $d_n(h)$ is the error in y_{n+1} resulting from one step of the method starting with the correct value $y(t_n)$. Note that this agrees with the definition given earlier for the explicit Euler method. However, it differs slightly from the definition given for the implicit Euler method. That difference is not important here.

Definition. The order of the general one-step method is the largest integer r for which

$$d_n(h) = O(h^{r+1}) \quad \text{as} \quad h \to 0.$$

This concept is in agreement with the discussion on the Euler methods, where the order was 1 and the local error went like h^2. A fact that is easy to see (and is left as an exercise) is that the condition of consistency is equivalent to the condition that the order is at least 1.

The next theorem tells us that the global errors satisfy a bound with one lower power of h than in the local errors. I.e. the convergence of

the method is displayed in a more explicit way than in the previous theorem. The proof is a direct imitation of what was done in Section 3(b).

Theorem (convergence): For a consistent general one-step method of order r, with $|d_n(h)| \le Dh^{r+1}$, the global errors e_n satisfy

$$|e_n| \le Dh^r \frac{e^{bL}-1}{L} + e^{bL}|e_0| .$$

Proof: From

$$y_{n+1} = y_n + h\psi(y_n, t_n, h)$$

and

$$e_{n+1} = y_{n+1} - y(t_{n+1}),$$

we get

$$\begin{aligned} e_{n+1} &= y_n + h\psi(y_n, t_n, h) - y(t_{n+1}) \\ &= y_n - y(t_n) + h\psi(y_n, t_n, h) + y(t_n) - y(t_{n+1}) \\ &= e_n + d_n + h[\psi(y_n, t_n, h) - \psi(y(t_n), t_n, h)] \\ |e_{n+1}| &\le |e_n| + |d_n| + hL|e_n| \\ &\le (1+hL)|e_n| + Dh^{r+1}. \end{aligned}$$

The rest of the proof is a matter of accumulating these inequalities, exactly as done before. QED

To determine the order of a given method, we generally insert an appropriate Taylor series into the formula defining the method, and determine the behavior of d_n as $h \to 0$. For the implicit case, this can mean a considerable amount of work. In any case, if the functions involved are all sufficiently smooth, we end up with a result of the form

$$d_n(h) = h^{r+1}\varphi(y(t_n), t_n) + O(h^{r+2}).$$

The function φ, which involves f,y, and their derivatives, but not h, is called the principal error function. We can then write $e_n = h^r \delta_n$ for the global errors, write a difference equation for δ_n, and so show that $\delta_n = \lim_{h \to 0} \delta(t_n)$, where $\delta(t)$ is a function defined by the ODE

$$\dot{\delta}(t) = f_y(y(t), t)\,\delta(t) + \varphi(y(t), t).$$

with initial value

$$\delta(0) = e_0 / h^r .$$

This generalizes the result derived for the explicit Euler method, where r=1 and $\varphi = -\ddot{y}/2$.

So far, we have assumed whatever smoothness we needed to get results conveniently. Consider now the possibility that such smoothness is lacking. We still assume f is Lipschitz in y and continuous in t, so that the basic existence theorem applies, and so that $\dot{y}$ exists and is continuous. Now look at the result of differentiating $\dot{y} = f$ once,

$$\ddot{y} = f_y \dot{y} + f_t .$$

If f_y, f_t exist and are continuous, then the same is true for $\ddot{y}$. We can continue this reasoning, and say that if f has continuous partial derivatives up to order q, then $y^{(q+1)}$ is continuous.

Now suppose we have a one-step method of order r, where we know $d_n = O(h^{r+1})$ and $e_n = O(h^r)$ for smooth problems. If we have q=r above (i.e. f has continuous partial derivatives up to order r), and if $\partial^r \psi / \partial h^r$ is also continuous, then the definition of $d_n(h)$ can be expanded in a Taylor series far enough to get $d_n = O(h^{r+1})$, and the convergence theorem

applies, giving $e_n = O(h^r)$. But if the maximal order q of continuous derivatives of f is less than r, then d_n will generally not be $O(h^{r+1})$, but only $O(h^{q+1})$, and we can only prove that $e_n = O(h^q)$. Thus in general, the maximum order of continuous partial derivatives of f and ψ is the maximum actual order attainable. This is only a general heuristic statement, to which there are exceptions. The text (p. 68) gives an example where q=1, but when a method of order r=2 is used, the global errors are $e_n = O(h^2)$.

The result which relates e_n to an ODE in $\delta(t)$ requires one further continuous derivative of f and ψ to exist, because the series for $d_n(h)$ is being carried one term further there.

So far, the step size h has been assumed to be constant throughout the problem. In practice, this is rarely true. Either the user changes h, or h is changed by a program with an automatic mechanism for doing this. It is therefore important to know that all of the results in this section can be obtained in the case of varying h as well. It is just somewhat harder and more involved.

Consider for example the convergence theorem, and write

$$t_{n+1} = t_n + h_n , \qquad h = \max h_n .$$

In the recursive inequalities for e_n, we simply replace h by h_n. Then in accumulating these, we will get bounds like

$$|e_n| \le \left[\prod_0^{n-1}(1+h_j L)\right]|e_0| + \frac{D h^r}{L}\left[\prod_0^{n-1}(1+h_j L) - 1\right]$$

(which reduces to the correct formula if all $h_n = h$). Then the inequality

$$\prod_0^{n-1}(1+h_j L) \le \prod_0^{n-1} e^{h_j L} = e^{\sum h_j L} \le e^{bL}$$

gives the same bound for e_n as in the earlier convergence theorem.

6. (b) An example

We consider the problem

$$\dot{y} = f = 2t - 1000\,(y - t^2)\,, \quad y(0) = 0\,, \quad 0 \le t \le 1\,.$$

It can be seen by inspection that the true solution is $y = t^2$. In fact, if we define $z = y - t^2$, the ODE becomes $\dot{z} = -1000z$, for which the general solution is $z = z_0 e^{-1000t}$, or $y = t^2 + z_0 e^{-1000t}$. The particular solution sought has $z_0 = 0$.

We consider here the numerical solutions obtained by the explicit and implicit Euler methods. The text (p. 17) gives the numerical results of the explicit method, and shows that reasonable accuracy is not obtained until the stepsize h is about .001 or less. When it is, the errors at $t = 1$ are about

$$e_n \approx -10^{-6} \qquad (h = 10^{-3})$$
$$e_n \approx -10^{-7} \qquad (h = 10^{-4})$$
$$e_n \approx -10^{-8} \qquad (h = 10^{-5})\,,$$

or, generally, $e_n \approx -10^{-3}\, h$ for any $h \le 10^{-3}$.

In the case of the implicit Euler method, we must solve the implicit equation

$$y_{n+1} = y_n + h\,\dot{y}_{n+1} = y_n + h\left[2t_{n+1} - 1000\,(y_{n+1} - t_{n+1}^2)\right]\,.$$

Because f is linear in y here, this is easily done:

$$y_{n+1} = \frac{y_n + h(2t_{n+1} + 1000t_{n+1}^2)}{1 + 1000h}\,.$$

For a given h (= a submultiple of 1) we take 1/h steps to get to t = 1. For example, for h = 1, we get, using $y_0 = 0$, $t_1 = 1$,

$$y_1 = \frac{0 + 1(2\cdot 1 + 1000\cdot 1^2)}{1 + 1000\cdot 1} = 1 + \frac{1}{1001} .$$

For h = 1/2, we find

$$y_1 = \frac{251}{1002} \left(\approx \frac{1}{4}\right), \qquad y_2 = 1 + \frac{251}{501\cdot 1002} .$$

Thus, the actual errors at t = 1 are

$$e_1 \approx 10^{-3} \qquad (h = 1)$$
$$e_2 \approx .5\cdot 10^{-3} \qquad (h = 1/2)$$

or, generalizing to arbitrary h,

$$e_n \approx 10^{-3}\, h .$$

These actual errors illustrate the fact that the theoretical error bounds obtained earlier are not always useful. For the explicit case, the bound of

$$|e_n| \leq \frac{Dh}{L} (e^{bL} - 1)$$

is useless since bL = L = 1000 here. Also, the similar bound for the implicit method cannot even be applied here, since it requires hL < 1, and this is far from the case for the values of h used above. Thus, these theoretical bounds are far more pessimistic than actual practice here.

Consider, however, the more sophisticated error estimates based on an auxiliary ODE. Here we write, for order 1,

$$e_n \approx h\, \delta(t_n) , \qquad \dot{\delta} = f_y \delta + \phi , \qquad \delta(0) = 0 .$$

Here $f_y = -1000$ and the prinicpal error function ϕ is $\pm\ddot{y}(t)/2 = \pm 1$ (-1 for the explicit case, + 1 for the implicit). The solution is seen to be

$$\delta(t) = \pm(1-e^{-1000t})/1000 .$$

Hence $\delta(1) \approx \pm 10^{-3}$, and the estimate of

$$e_n \approx \pm 10^{-3} h$$

is in complete agreement with the calculated results.

Suppose, instead of solving for the implicit Euler solution exactly, we had tried an iterative method. If we consider functional iteration, we predict $y_{n+1(o)}$ for the value of y_{n+1} and use

$$y_{n+1(m+1)} = y_n + hf(y_{n+1(m)}, t_{n+1}) , \qquad m = 0, 1, \ldots .$$

This leads to badly divergent answers for h = 1 or h = 1/2. The reason can be seen by looking at

$$\delta_{(m)} = y_{n+1(m)} - y_{n+1}$$

and subtracting the relation

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1})$$

to get

$$\delta_{(m+1)} = -1000h\ \delta_{(m)}$$

for this case. Thus, if $h > .001$ and $y_{n+1(o)}$ is at all in error (i.e., $\delta_{(o)} \neq 0$), then $\delta_{(m)}$ diverges as $m \to \infty$. This divergence is worse the larger h is. This behavior is <u>not</u> a result of the use of the explicit Euler method to get $y_{n+1(o)}$. Even if the true solution t^2_{n+1} is taken as the predicted value, the method diverges, as long as $h > .001$. Conversely, if $h < .001$, this iteration method will converge

to the desired solution regardless of the prediction. (It is true, however, that this convergence failure and the instability of the explicit Euler method both have the same cause — the large size of f_y.)

We might alternatively have chosen Newton's method instead of functional iteration. Here, because the equation in y_{n+1} to be solved is linear, and Newton's method is based on linearizing that equation, we would get the correct y_{n+1} in one iteration.

In short, the implicit Euler method is quite successful for this problem, but its success relies on the use of a sufficiently good method of solving the resulting implicit equation. The fact that functional iteration is a bad choice is due to the problem, and should not be blamed on the basic method (implicit Euler). It is a matter of the implementation of the method. Some problems will require a more powerful implementation of a given method than others, and this example is one that requires something more powerful than functional iteration for the implicit Euler method.

Knowing the results of the numerical solutions from the two methods, we can compare them, and ask, "Which method is better?" The fact that a one-step solution gives a good answer with the implicit method, and not with the explicit one, suggests that the implicit one is better. If we will accept an error of $\pm 10^{-3}$, it is. In fact, if we require less than six decimal places of accuracy at $t = 1$, then we can obtain the desired accuracy with the implicit method with many fewer steps than the explicit one. The latter requires about 1000 steps before even one decimal place is obtained, and the former requires as little as one step.

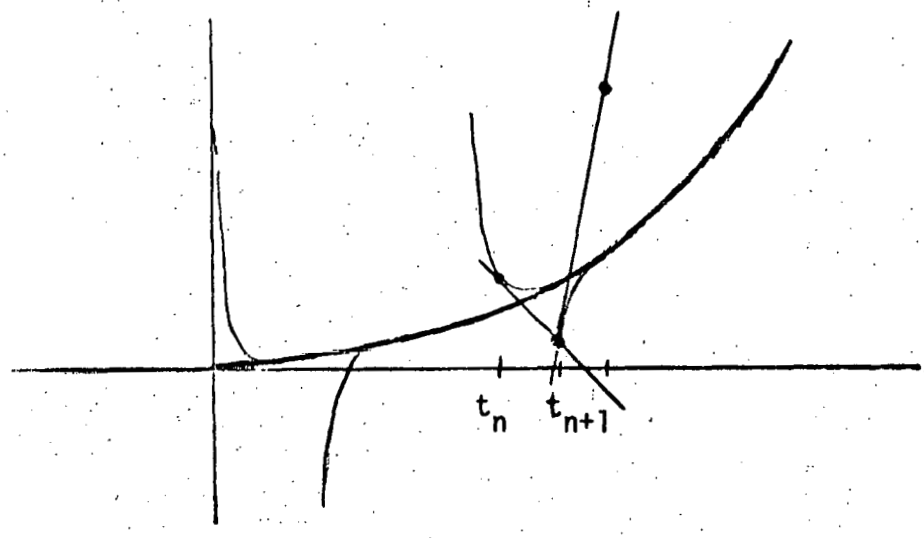
But to say that the implicit method is always the better one for this problem would be a mistake. We know that at $t = 1$, $|e_n|$ is asymptotically about $10^{-3}h$ for either method, and so the asymptotic errors give no basis for a choice. The hitch is that the actual $|e_n|$ agree for the two methods only when h is about .001

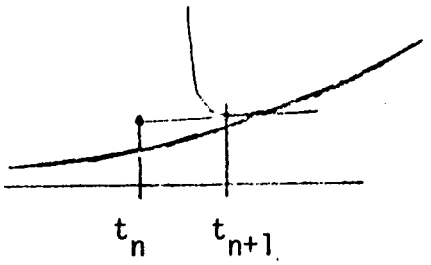
or less, and then we get at least six places of accuracy. Hence, if the desired accuracy is six or more places, either method suffices, and we must use considerations other than the number of steps (i.e., the size of h) to make a choice. The only remaining criterion is the efficiency of performing each step. In this respect, the explicit method is somewhat better, as it requires 4 multiplies per step, vs. 5 multiplies and a divide (which could be eliminated, however) for the implicit method. This difference is not great, but if total computational cost were of paramount importance, the explicit method would be the better choice, for 6-place accuracy or better.

The relative inferiority of the explicit method under low accuracy requirements can be illustrated graphically. We first draw the curves for various particular solutions (various z_o). These approach the curve $y = t^2$ very rapidly, and then become (graphically) identical to that curve. If the explicit Euler method has been applied up to a certain point (t_n, y_n), where y_n has only a small error, the next step, taken according to

$$y_{n+1} = y_n + h\dot{y}_n \quad ,$$

carries the solution to (t_{n+1}, y_{n+1}) along a line tangent to a solution curve at (t_n, y_n). This tangent is so steep, for all but very small errors in y_n, that y_{n+1} is then quite inaccurate, for all but very small values of h.





In the implicit case, however, the next point is on a line that is tangent to a solution curve at t_{n+1}. Even for large h and/or sizable errors in y_n, the resulting y_{n+1} will not be drastically in error, by the nature of that tangent. An inspection of the graphical construction will reveal that a large error in y_{n+1} would correspond to a steep tangent line that could not possibly pass through (t_n, y_n) because it slopes the wrong way.

This example may appear quite contrived, and carefully designed to produce the behavior just discussed. It <u>is</u> an artificial problem, of course, contrived to make the numerical solution rather simple to obtain and analyze. However, the resulting behavior is <u>not</u> contrived, but typical of large classes of problems This is a simple example of a <u>stiff</u> problem, and stiffness is a rather common property of ODE's that arise in many areas of applications. Qualitatively, stiff problems are characterized by the presence of <u>rapid transient</u> solutions (given by the $z_0 e^{-1000t}$ term here), together with relatively <u>smooth</u> long-range or <u>equilibrium</u> solutions (given by $y = t^2$ here). In numerical solutions, the main symptom of stiffness is that some methods require very much smaller values of h than others, for a given accuracy. In this example, if we were to change the ODE to

$$\dot{y} = 2t - 10^6(y - t^2) \quad ,$$

it would be even stiffer, and would require an h of about 10^{-6} or less for the explicit Euler method, while the implicit one gives 6-place accuracy with only h = 1. The fact that the problem is contrived to be linear and very simple has nothing to do with this, but simply makes the difficulties of numerical solution

more transparent. The use of these two particular solution methods here is also not contrived to produce these results. There are large classes of methods, in common usage, which will have the same difficulty on this problem as the explicit Euler method. And there are classes of methods which overcome that difficulty in the same way that the implicit Euler method does.

The problem of stiffness, and the explanation of its effects on different methods, will be covered more fully later (Part III).

Exercise 6.1

Consider the trapezoid rule, a one-step implicit method:

$$y_{n+1} = y_n + \tfrac{1}{2} h (\dot{y}_n + \dot{y}_{n+1}).$$

(a) Using Taylor series, determine the order of the method, and the leading term in the local error

$$d_n = y(t_n) + \tfrac{1}{2} h \left[f(y(t_n), t_n) + f(y(t_{n+1}), t_{n+1}) \right] - y(t_{n+1})$$

(Disregard the discrepancy between this d_n and that of the general definition.)

(b) What is the absolute stability region,

$$S_a = \{ h\lambda : |y_{n+1}| \leq |y_n| \text{ for } \dot{y} = \lambda y \} \ ?$$

(c) For the problem $\dot{y} = 2t - 1000(y - t^2)$, $y(0)=0$, $0 \leq t \leq 1$, h arbitrary, what is the global error e_n (ignoring roundoff and iteration error)?

Exercise 6.2

Given a general one-step method with local truncation error

$$d_n(h) = h^{r+1} \varphi(y_n, t_n) + O(h^{r+2}), \quad \varphi \not\equiv 0,$$

show that the method is consistent if and only if the order r is ≥ 1.

7. Implementation

This section covers a number of topics related to the implementation of typical ODE methods into practical algorithms and computer programs. In the course of dealing with these topics, it will become clear that some methods are easier to implement than others, in various respects. Considerations of this kind, together with considerations related to the theoretical properties of the methods, must be used if the best available method is to be chosen for a given problem. That best method depends on the problem as well. No one method or no one implementation of a method is best for all problems. Properties of the problem must be known, either in advance of a numerical solution, or as a result of trying such a solution.

The various considerations, relative to both theoretical properties and matters of implementation, cannot be given completely prior to a more detailed discussion of the methods under consideration. In particular, the Runge-Kutta methods and the linear multistep methods will be studied more closely later. For the present, however, a number of practical matters are covered in a general setting, in what follows.

7. (a) History vector; storage

For any method, some information must be saved from step to step to implement the method meaningfully. This fact is particularly evident for the multistep methods, typified by the linear multistep methods:

$$y_n = \sum_{i=1}^{K_1} \alpha_i y_{n-i} + h \sum_{i=0}^{K_2} \beta_i \dot{y}_{n-i} .$$

In going from y_{n-1} to y_n, one must have, in some form, the data

$$y_{n-i} \qquad (1 \le i \le K_1)$$

$$\dot{y}_{n-i} \qquad (1 \le i \le K_2).$$

This comprises $L = K_1 + K_2$ pieces of data, which can be simply put together to make up a history vector of length L, with vectors of length N as components. This is then really a history array, of size LN. Rather than store this data as written above, it is often desirable to store some set of linear combinations of it. But any such alternative history vector must still contain the same number of quantities, LN.

This storage problem is often regarded as a drawback of the multistep methods, especially when N is large. L can also be large; some higher order methods in use require values of L up to 20. Thus for a large system, say with N = 1000, a storage requirement of 20,000 locations might be too much of a drawback for the use of the method in question. It might even be prohibitive, if the core memory available is insufficient.

On the other hand, there may well have been a strong reason for choosing the method with large L, and it may be impossible or undesirable to reduce N. Hence rather than sacrifice the size of the problem or the order of the method, the user may be forced to deal with the problem of storing the history array directly, such as by way of extended storage devices.

7. (b) Solution of the implicit equation

Any implicit method is more difficult to implement than a similar explicit one, because it requires an accompanying method for the solution of nonlinear systems of algebraic equations. That problem, while on a lower mathematical level than the ODE problem, is still not an easy one. Presum-

ably, the user has a strong motivation to use an implicit method in the first place, counterbalancing this extra difficulty. The example $\dot{y} = 2t - 1000(y - t^2)$, $y(0) = 0$, as solved by the explicit and implicit Euler methods, illustrates such a motivation.

To be specific, consider the implicit linear multistep methods. Here the implicit equation for y_n is

$$y_n = h\beta_0 f(y_n, t_n) + a_n$$

where $\beta_0 \neq 0$ and a_n is a known vector containing past history. (For the implicit Runge-Kutta methods, the other basic method class in which the implicit equation problem occurs, the equation will have much the same form, in terms of the unknown intermediate value or values.) Usually h is small, and we can often take advantage of the smallness of h and of the particular form of the equation.

As we are not assuming any special form of f or any information about it other than the Lipschitz condition, we must use iterative methods to solve the equation. No direct method is available except in very special cases. When we use an indirect or iterative method, we must start with a guess or prediction, $y_{n(0)}$, and then iterate by some rule to get corrections $y_{n(m)}$ ($m = 1,2,\ldots$) which hopefully converge to the true solution y_n of the equation. The term predictor-corrector method is often applied to this process, especially in the linear multistep context. An implementation of the method will require some rule for stopping the iterations, and accepting the last iterate as y_n. Such rules are usually based on a test of the successive iterate difference, $y_{n(m+1)} - y_{n(m)}$.

The simplest of the iterative methods is functional iteration, in which the function of y_n on the right above is iteratively evaluated. Dropping

the subscript n, which is fixed throughout this discussion, the iterates $y_{(m)}$ are then given by

$$y_{(m+1)} = \beta_0 h f(y_{(m)}, t_n) + a_n .$$

This method was illustrated for a simple example earlier, and we can analyze the convergence in general almost as easily. Consider the iteration error,

$$\delta_{(m)} = y_{(m)} - y_n .$$

A subtraction then gives

$$\delta_{(m+1)} = h\beta_0 \left[f(y_{(m)}, t_n) - f(y_n, t_n) \right]$$

$$|\delta_{(m+1)}| \leq |\beta_0| h L |\delta_{(m)}| .$$

If L is not unusually large, it is reasonable to expect that $|\beta_0|$ hL < 1 for the values of h being used. If so, then we can conclude that

$$\delta_{(m)} \longrightarrow 0 \qquad \text{as} \qquad m \longrightarrow \infty ,$$

or that we have <u>convergence</u> of the correctors. I. e., the iteration error can be made arbitrarily small by taking enough iterations. Moreover, the constant $|\beta_0|$ hL is a measure of the rapidity of convergence — the smaller it is, the fewer iterations are needed. (This convergence must be distinguished from convergence of the basic method, which deals with the total numerical solution as h → 0. Here only one step is being discussed and h is fixed.)

There are frequently problems where L is so large that the requirement $h < 1/|\beta_0|$ L makes this process too costly. This does <u>not</u> mean that the choice of the implicit method was bad, only that the choice of corrector iteration method used to <u>implement</u> it was a poor one.

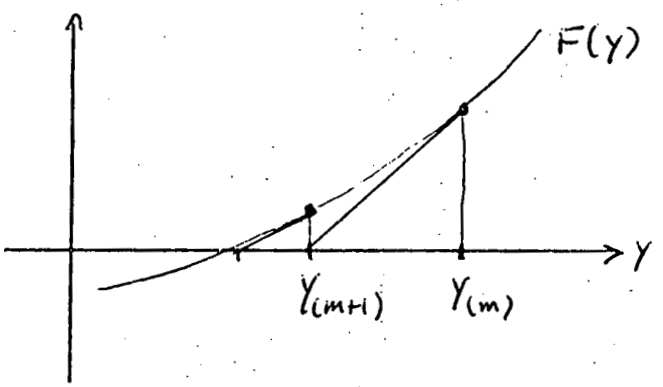
Another logical choice of solution method for the implicit equation is Newton's method. Here we do not really take advantage of the known form of the equation, but simply write it as

$$F(y) \equiv y - \beta_0 h f(y, t_n) - a_n = 0.$$

To apply Newton's method in the scalar (N = 1) case, we guess $y_{(0)}$ and use

$$y_{(m+1)} = y_{(m)} - F(y_{(m)}) / F'(y_{(m)})$$

to iterate. Graphically, this corresponds to drawing a tangent to the curve F(y) at $y_{(m)}$ and following it down to the y axis at $y_{(m+1)}$. Repetition of the process, under suitable conditions, will lead to the desired root y_n.



The generalization of this to the vector case N > 1, where F(y) is a vector-valued function of a vector, is given by the formula

$$y_{(m+1)} = y_{(m)} - \left[F_y (y_{(m)}) \right]^{-1} F(y_{(m)}).$$

Here F_y is the N x N Jacobian matrix of F:

$$F_y = \frac{\partial F}{\partial y} = \left(\frac{\partial F^i}{\partial y^j} \right)_{i,j=1}^{N}.$$

The notation here is to write y as the vector

$$y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{pmatrix} = (y^i)_{i=1}^N$$

and similarly for F, with components

$$F^i(y) = F^i(y^1, y^2, \cdots, y^N).$$

We then form a matrix of partial derivatives

$$P = (P_{ij})_{i,j=1}^N = F_y,$$

$$P_{ij} = \frac{\partial F^i(y)}{\partial y^j},$$

where p_{ij} is the element of P in row i and column j. We evaluate P (i.e., the p_{ij}) at the vector $y_{(m)}$, and then we want the vector

$$y_{(m+1)} = y_{(m)} - P^{-1} F(y_{(m)}).$$

Here P^{-1} is another N x N matrix, with the property that

$$(P^{-1}) \cdot P = P \cdot (P^{-1}) = I,$$

where we do the standard multiplication of matrices and I is the standard N x N identity matrix

$$I = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}.$$

Then $P^{-1}F$ is the usual product of a matrix and a vector.

There is an alternate way to view these relations, which is actually preferred for practical reasons. That is to write the iteration formula as

$$P \cdot \left[y_{(m+1)} - y_{(m)} \right] = -F(y_{(m)}).$$

This is to say that the vector

$$x = (x^i)_{i=1}^{N} = y_{(m+1)} - y_{(m)}$$

is the solution of the system of linear equations

$$Px = -F(y_{(m)}).$$

The individual equations are

$$\sum_{j=1}^{N} p_{ij} x^j = -F^i(y_{(m)}) \quad (1 \le i \le N).$$

In practice we can thus call upon a linear system solver, for which numerous techniques and computer subroutines exist, apply it for given coefficient matrix P and right-hand side vector $-F(y_{(m)})$, and get a result x, for which

$$y_{(m+1)} = y_{(m)} + x$$

is the next corrector iterate. In doing so, the matrix P^{-1} never appears as such at all.

Recalling what F(y) was, we may write its Jacobian by differentiating each term separately. The Jacobian (with respect to y) of y is I, and that of a constant is the zero matrix. Hence

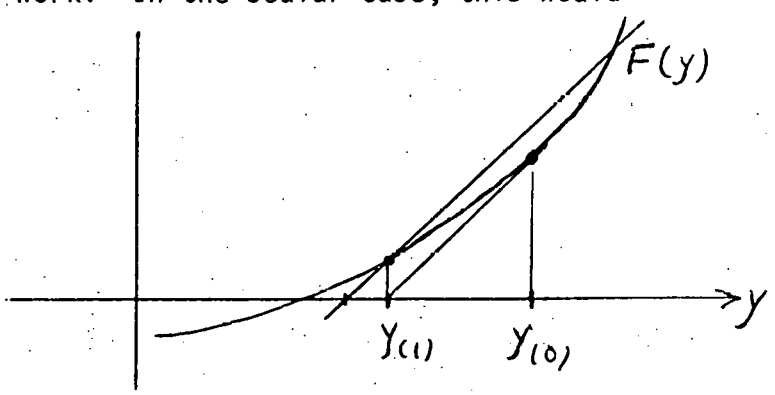
$$P = F_y = I - \beta_0 h f_y = I - \beta_0 h J,$$

where

$$J = f_y = \frac{\partial f}{\partial y} = \left(\frac{\partial f^i}{\partial y^j} \right)$$

is the Jacobian of the right-hand side of the given ODE system. J is often referred to simply as <u>the</u> Jacobian of the problem.

J can be rather costly to evaluate and manipulate, especially if N is large. If we take Newton's method in its strict form, a great deal of effort will go toward simply evaluating $P = F_y(y_{(m)})$, which must be done at every iteration on every step. For efficiency's sake it would be much better not to have to spend that much effort, and fortunately, it is not necessary to. We can, for example, simply use $P = F_y(y_{(0)})$, which is the same for all m, and hence eliminate considerable work. In the scalar case, this would mean using the slope of the tangent at $y_{(0)}$ again at $y_{(1)}$, etc. instead of recomputing the slopes. For all but the first step, the line used is thus not a tangent to the curve, but a <u>chord</u> through it. Hence this variation of Newton's method, often called a <u>quasi-</u>



Newton method, is also called a chord method.

There are even further departures from Newton's method which also work quite well in practice. We might, for example, use a matrix P on step n that was evaluated at some earlier step, say step n' (n' < n):

$$P = F_y(y_{n',(c)})$$

This would save us the effort not only of recalculating F_y at every iteration, but also of recalculating it at each step. Other possibilities arise if we only approximate F_y in some way, such as by using finite differences in place of the partial derivatives, or by using some even more crude approximation.

The success of Newton's method will naturally be somewhat impaired by switching to a chord method. However, under suitable conditions on F, it is possible to prove that the iterates $y_{(m)}$ still converge to y_n, though possibly not as rapidly. Such a proof is beyond our scope here.

The fact that F_y is only being approximated, possibly with very little accuracy, does not mean we are sacrificing any accuracy in the final corrector approximation to y_n. That is, if we stop the iterations at $y_{(M)}$ and accept this as our approximation to y_n, we can still make the iteration error $y_{(M)} - y_n$ as small as we like, by proper choice of M. The inaccuracy in P affects only the rate of convergence to y_n, not the final answer.

The choice of the initial prediction $y_{(0)}$ is usually a crucial detail here, regardless of corrector method. For a given method, $y_{(0)}$ must not be too distant from y_n, or the iterates will fail to converge. The chord methods are especially sensitive to the accuracy of $y_{(0)}$, although they make up for that by being more rapidly convergent in general than functional iteration.

For a linear multistep method, one must use the available history information, with data at $t_{n-1}, t_{n-2}, \ldots$, to extrapolate to t_n, in order to get

an accurate prediction. This usually amounts to using the appropriate explicit method based on the same history vector.

7. (c) Error estimation and choice of step size and order

This section deals with the difficult but very practical matter of error control. The control of errors involves first the estimation of errors, to some extent, and then the practical selection of step sizes h, and (if methods of several orders are provided) of the order r as well.

To begin with, consider the numerical solution of a given problem over a given finite interval, with a fixed method and fixed h. It would be natural to require of the calculation that the global error e_n at the end of the problem be bounded by some constant E, which the user specifies. If the method selected has order r, then we know from the appropriate convergence theorem that

$$|e_n| \leq K h^r$$

for some constant K, provided h is sufficiently small. By definition of r, there is no larger integer for which that bound holds in general. If the problem is sufficiently smooth, we can be more precise by deriving an auxiliary ODE for a function $\delta(t)$ for which, when h is sufficiently small,

$$e_n = h^r \delta(t_n) + O(h^{r+1}).$$

In any case, we can expect that $|e_n|$ will go roughly like h^r, as we vary h within bounds that produce reasonably accurate answers at all.

It follows that if we consider different values of the error bound E and corresponding values of acceptable step sizes h, h will be roughly proportional to $E^{1/r}$. To make practical use of this fact, we might, for example, make one run with some value of h which seems reasonable, getting

a final answer Y_A, and then make a second run with step size h/2, getting an answer Y_B. Then, on the assumption that the true final answer Y satisfies

$$Y_A - Y \approx h^r \delta \quad (|\delta| = K),$$

$$Y_B - Y \approx (h/2)^r \delta,$$

we obtain $|Y_A - Y_B| \approx (1 - 2^{-r})Kh^r$. Hence, the error in the (presumably more accurate) answer Y_B can be estimated as

$$|Y_B - Y| \approx 2^{-r}|Y_A - Y_B|/(1-2^{-r}) \equiv E_B.$$

Moreover, if this error is not acceptable, any further runs, with stepsize h' and answer Y_C, will have a predictable final error of

$$E_C \approx \left(\frac{h'}{h/2}\right)^r E_B.$$

As we want E_C to be roughly equal to a given bound E, we could even determine the correct value of h':

$$h' = (E/E_B)^{1/r} \, h/2.$$

Suppose now that several different orders are available. For example, a program might contain the explicit Adams methods of orders 1 to 4, and any r in that range could be selected. For each r, there will be a constant K_r for which the final error is roughly $K_r h^r$ for small enough h. The values of the K_r might even be known from theoretical properties of the methods and the problem.

The selection of the best order will have to involve efficiency considerations also. We can uaually assign a cost per step k_r to the method of order r. This might simply be the number of f evaluations, for a crude approximation. Then if the rth order method is used with stepsize h_r, and the interval length is b, the total numbers of steps is

$$N_r = b/h_r ,$$

and the total work for the problem is the sum of the costs per step,

$$W_r = N_r k_r = k_r b/h_r .$$

The norm of final error, e, can then be plotted as a function of W_r^{-1} according to

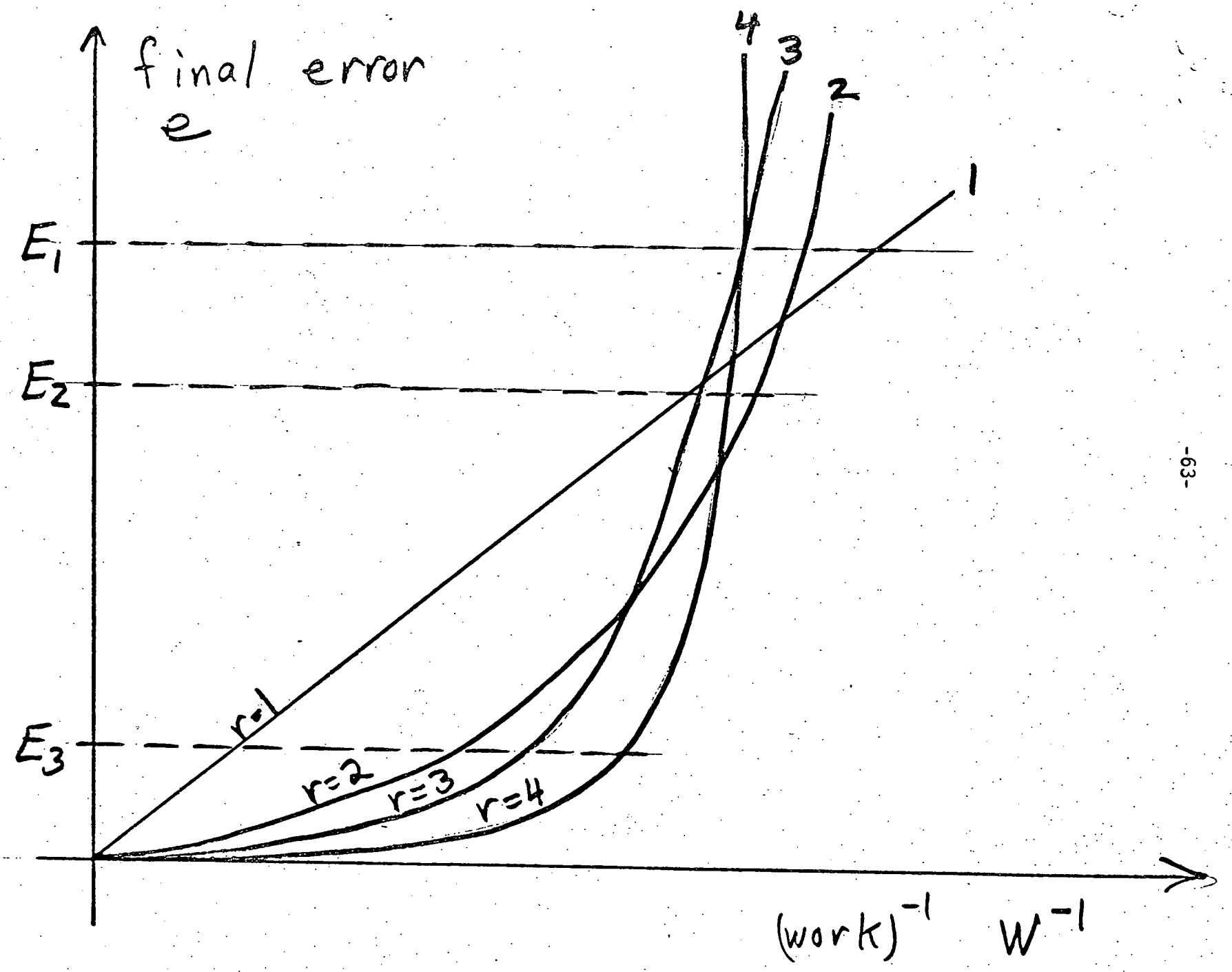
$$e \approx K_r h_r^r = K_r (k_r b/W_r)^r$$

$$= A_r (W_r^{-1})^r ,$$

$$A_r = K_r (k_r b)^r .$$

(See the accompanying graph, and also p. 75 in the text, where a generalization to variable step sizes is discussed.)

We are interested in achieving e = E, and we of course, want to minimize W_r. Thus the right choice is the r for which W_r^{-1} is largest along the line e = E. For different values of E we can get different optimal orders. If E is quite small, clearly the curves for different r descend vertically


final error
e
4
3
2
1
E_1
E_2
E_3
r=1
r=2
r=3
r=4
(work)$^{-1}$ W^{-1}

as r increases, so that the maximum r is the optimal choice. However, for larger E, lower orders may be optimal. No general statement can be made except in the limit of small E. The higher order methods are not invariably the most efficient for given accuracy. For a given h, or a given total work W, a higher order does not invariably give better accuracy. The absence of general rules here is all the more assured by the fact that these simple curves themselves do not apply above certain levels in E or h.

The goal of bounding the final global error, while a natural and desirable goal, is not a practical one. When we must select h and possibly also r at the beginning of the problem, and possibly also during the solution of the problem, the information we would need for those selections, if a final error bound is to be met, is simply not available in most cases. That information involves the unknown solution at future values of t, and it may also involve decisions about h and r that are to be made in the future.

An alternative that is quite feasible is to bound the norm of the local error only. That is we can make selections of h and r which satisfy $|d_n| \leq \varepsilon$, where ε is a prescribed tolerance. This strategy is at the other extreme from the earlier one, and there are intermediate choices. For example a bound $\varepsilon(h,t)$ on $|d_n|$ might be devised that varies during the problem, in an attempt to achieve a global error bound. As this would complicate the problem of error control considerably, we will consider only case of constant ε, which itself can be rather complicated.

The first and most difficult task in controlling local error is simply to estimate it. Of course we need not estimate errors as accurately as we estimate the solution values to the original problem, but some kind of an estimate is necessary.

A standard method of estimating d_n, used for one-step methods, is the method of step doubling. Here we use the fact that $d_n(h) \approx h^{r+1}\phi(t_n)$ where ϕ is a function (usually a combination of derivatives of f) which is not known in advance but is independent of h. Consider taking two successive steps of size h, from t_n to t_{n+1} to t_{n+2}. If y_n was considered exact, then the calculated values y_{n+1} and y_{n+2} will have errors (respectively) of

$$e_{n+1} = d_n(h)$$
$$e_{n+2} \approx 2d_n(h).$$

Here e_{n+2} is approximated by a simple accumulation of the local errors, discarding terms that are $O(h^{r+2})$ that were kept in the rigorous global analysis. On the other hand, we may consider doubling h and taking a single step from t_n to t_{n+2}. We then get a calculated value $\bar{y}_{n+2}$ which has an error

$$\bar{e}_{n+2} = d_n(2h).$$

It follows that

$$\Delta \equiv \bar{y}_{n+2} - y_{n+2} = \bar{e}_{n+2} - e_{n+2}$$
$$\approx d_n(2h) - 2d_n(h)$$
$$\approx \left[(2h)^{r+1} - 2h^{r+1}\right]\varphi(t_n)$$

and hence that, within an error of $O(h^{r+2})$,

$$\Delta \approx 2(2^r - 1)\, d_n(h),$$

and so the local errors in y_{n+2} and $\bar{y}_{n+2}$ may be estimated:

$$e_{n+2} \approx \Delta/(2^r - 1)$$
$$\bar{e}_{n+2} \approx 2^r \Delta/(2^r - 1).$$

The one-step local error $d_n(h)$ would then be estimated as $\frac{1}{2}\Delta/(2^r - 1)$.

In the case of multistep methods, step doubling is generally not used, because much less expensive error estimates are available, using the more extensive history of the calculation.

For the general linear multistep method,

$$y_n = \sum_{i=1}^{K_1} \alpha_i y_{n-i} + h \sum_{i=0}^{K_2} \beta_i \dot{y}_{n-i} ,$$

the local truncation error $d_n = d_n(h)$ is defined as the difference between the left and right sides of the above equation when y_k is replaced by $y(t_k)$ and $\dot{y}_k$ by $\dot{y}(t_k)$ for a true solution $y(t)$ of the ODE. (A more detailed discussion of this will be given later.) We will see later that if the method is of order r, then we have an asymptotic formula (for small enough h)

$$d_n = C h^{r+1} y_n^{(r+1)} + O(h^{r+2})$$

for some constant C. So we estimate d_n by estimating the current value of $y^{(r+1)}$ by use of a linear combination of the history data. For example, in the explicit Euler method, where $r = 1$ and $C = -\frac{1}{2}$, we can take

$$-\frac{h\dot{y}_n - h\dot{y}_{n-1}}{2} \approx -\frac{h^2 \ddot{y}_n}{2} \approx d_n .$$

For the implicit linear multistep methods, a general technique for error estimation is the use of the predictor-corrector difference. By this is meant the difference between a predictor $y_{n(0)}$ that is also of order r and the final corrected solution y_n of the implicit equation. Because the predictor has order r, we can write

$$y_{n(0)} - y(t_n) = C' h^{r+1} y^{(r+1)}(t_n) + O(h^{r+2})$$

while the error in y_n is

$$y_n - y(t_n) = d_n + O(h^{r+2})$$
$$= Ch^{r+1} y^{(r+1)}(t_n) + O(h^{r+2}).$$

Hence the difference is

$$y_n - y_{n(0)} = (C - C')h^{r+1} y^{(r+1)}(t_n) + O(h^{r+2}).$$

By ignoring all $O(h^{r+2})$ terms, we arrive at an estimate for d_n in the form

$$d_n \approx \left(\frac{C}{C - C'}\right)[y_n - y_{n(0)}].$$

There is some theoretical doubt as to what conditions are necessary to give this estimate a rigorous foundation — as to whether the solution curve y(t) must go through $y_{n-1}, y_{n-2}, \ldots, y_{n-k}$ (for a k-step method), or whether the predictor and corrector formulas (i.e., the explicit formula for $y_{n(0)}$ and the given implicit formula for y_n) must satisfy certain coefficient conditions. But the idea seems to work well in practice. It is easy to use as both y_n and $y_{n(0)}$ must eventually be available anyway, and the constant $\frac{C}{C-C'}$ is known in advance from the formulas used. There is very little extra expense involved — no extra function evaluations, for example, as there is for the step doubling method.

Now suppose that, by whatever method is appropriate, an estimate of d_n is computed during the numerical solution. Let us call that estimate d_n also, and assume that we wish our choice of h and r to satisfy the test $|d_n| \leq \varepsilon$ for a prescribed tolerance ε. We now consider how to achieve that control.

Take first the case in which the order r is fixed, and the choice of h is the only decision to be made. We can first of all compute $|d_n|$, make the test against ε, and, if it is passed, do nothing to h and proceed. If it fails, we want to choose a smaller value of h, say h', for which the test will be passed. Recalling to the statement that d_n is roughly proportional to h^{r+1}, we have

$$\frac{|d_n(h')|}{|d_n(h)|} \approx \left(\frac{h'}{h}\right)^{r+1}.$$

We know $d_n = d_n(h)$ for the stepsize h currently being used, and we want $|d_n(h')| \leq \varepsilon$. Hence we conclude that we should make

$$h' \leq h\left(\frac{\varepsilon}{|d_n|}\right)^{\frac{1}{r+1}}.$$

If the approximations used are good, then any h' satisfying this inequality will lead to an acceptable local error. But we may as well maximize h', i.e., maximize the efficiency of the calculation, within the limits imposed by the local error test. Hence we would want to choose h' about equal to the above bound, possibly slightly less to allow for errors in estimates used:

$$h' \approx h\left(\varepsilon / |d_n|\right)^{\frac{1}{r+1}}$$

The reasoning above suggests that we might alter h even when the error test is passed, for a considerably larger (and hence preferrable) stepsize might also pass the test. It follows that the above formula for selection of h' could be used generally, regardless of the outcome of the test.

If the error test fails at any time, and the new stepsize h' is chosen accordingly, there is still another decision one must make. Either the

current step (taken with stepsize h) is repeated with h' before proceeding, or it is accepted and only the subsequent steps are taken with h'. The latter strategy is obviously less costly in the short run, but possibly more damaging in the long run, especially if the step is very much larger than is indicated by the error control.

The control of errors is complicated slightly if we must also select from a group of methods of several different orders. If the calculation is currently proceeding with the order r method and a stepsize of h, we will then wish, at least occasionally, to choose a (possibly) new order r' as well as a new stepsize h'.

Here the local truncation error d_n is a function of both the stepsize and order. At the current stepsize h but arbitrary order q, we have a function

$$d_n(h,q) = C(q)\, h^{q+1} y_n^{(q+1)} + O(h^{q+2}).$$

What was discussed earlier was the estimation of this quantity when q = r. When q < r, it is equally easy to estimate because the order of the derivative is lower and so less information would be needed to approximate it. When q > r, more information would be required than is normally saved to perform the rth order method. Hence some extra quantities must be saved to do this. For this reason, variable order codes usually restrict consideration to order r + 1 if the current order is r. This will be discussed at greater length later. In any case, assume that we have estimators of $d_n(h,q)$ for the various values of q under consideration. Then an approximate new stepsize that would be optimal at order q would be

$$h_q \simeq h\left(\varepsilon / |d_n(h,q)|\right)^{\frac{1}{q+1}},$$

on the basis of the local error test. This is arrived at exactly as before. Now we simply want to take the largest step possible within the limitations of the error test. So we compute

$$h' = h_{r'} = \max_{q}(h_q),$$

the largest of the different allowed stepsizes. If this maximum occurs at q = r' and we switch to the method of order r', then h' is the optimal stepsize we would use.

There may be other considerations that enter into this selection algorithm. We must still decide whether or not to redo the current step if it failed the test. Considerations of overhead costs may be used to bias the selection in certain ways. Finally, stability considerations may dictate that we not allow changes of h or r too frequently, or too drastically.

Exercise 7.1

Consider the second-order equation $\ddot{u} = -u$, on $0 \le t \le \pi/4$ with initial condition $u_0 = 0$, $\dot{u}_0 = 1$ (solution is $u = \sin t$).

(a) Convert this problem to one for a first-order system in a vector $y = \begin{pmatrix} y^1 \\ y^2 \end{pmatrix}$.

(b) Using the maximum norm on vectors ($|v| = \max |v^i|$) what is the best Lipschitz constant L?

(c) Among the explicit Euler, implicit Euler, and trapezoid rule methods, which would you choose and why? For a maximum error of $\pm 10^{-5}$, what h would you use? Use local and global error estimates, the L from (b), and initial error $e_0 = 0$, and assume $|u| \le 1$, $|\dot{u}| \le 1$, etc., for all calculated solutions.

(d) Perform one step of the trapezoid rule with $h = \pi/4$, solving the implicit equation exactly. What is the resulting estimate of $u(\pi/4)$?

Exercise 7.2

Consider a chord method for corrector iteration in an implicit linear multistep method. What happens if the P matrix (the coefficient matrix in the linear system) is taken to be the identity matrix I? What is the approximation to the Jacobian J that corresponds to this choice of P?

Exercise 7.3

Write the implicit Euler method as $y_n = y_{n-1} + h\dot{y}_n$, and consider the explicit Euler method as a predictor. If y_n is the true solution of the implicit equation, show that $\delta \equiv y_n - y_{n(o)}$ is $O(h^2)$. Calculate its leading term and derive an estimate of the local error d_n based on δ.

Part II: Study of Discrete Methods

8. Runge Kutta methods

The Runge-Kutta methods for $\dot{y} = f(y,t)$ constitute a large class of commonly used methods. They are one-step discrete methods, and so are subject to the analysis of general one-step methods done in Chapter 6, where the general form

$$y_{n+1} = y_n + h\psi(y_n, t_n, h)$$

was assumed. For the Runge Kutta methods, ψ has a particular structure, which is characterized by the use of intermediate t values (i.e. intermediate between t_n and $t_{n+1}=t_n+h$) in the required evaluations of f. The more commonly used Runge-Kutta methods are explicit, but a subclass of implicit ones also exists.

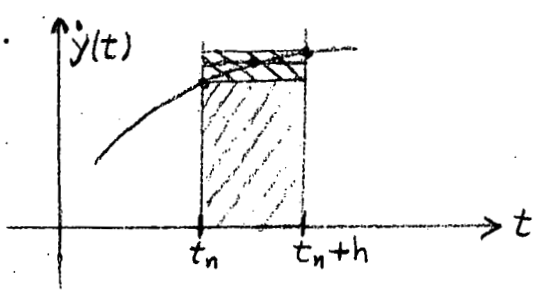
8. (a) Explicit 2-stage

To show how one can arrive at Runge-Kutta methods in a logical and simple-minded way, we return momentarily to the Euler methods. There we write

$$\Delta y_n \equiv y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_n+h} \dot{y}(t)\,dt,$$

and we replace the integral by either $h\dot{y}_n$ or $h\dot{y}_{n+1}$. In this sense, the Euler methods can be regarded as arising from quadrature methods, i.e. methods of approximating definite integrals in one dimension. Many (but not all) discrete ODE methods (and some nondiscrete ones) can be derived from quadrature methods in this way.

Graphically, in the case of a scalar ODE, the area under the graph of $\dot{y}(t)$ between $t=t_n$ and $t=t_n+h$ is being



approximated by the area of a rectangle of width h and height $\dot{y}_n$ or $\dot{y}_{n+1}$. It is intuitively clear that a much better approximation is obtained if we use a height of $\dot{y}(t_n+\frac{h}{2})$, i.e. the height at the midpoint of the interval. The quadrature method arising from this choice is the midpoint rule. Applying it to the integral for Δy_n yields the formula

$$y_{n+1} = y_n + h f\left(y(t_n+h/2),\, t_n+h/2\right).$$

The trouble is that the value of $y(t_n+h/2)$ required is not known. However, we can certainly approximate it with, for example, the explicit Euler method:

$$y(t_n+h/2) \approx y_n + \frac{h}{2}\dot{y}_n .$$

If we do this, we end up with the following algorithm for ODE solution, which is also called the midpoint rule:

$$\begin{cases} g_1 = y_n + \frac{h}{2} f(y_n, t_n) \\ y_{n+1} = y_n + h f(g_1, t_n+h/2) . \end{cases}$$

Notice that this is an explicit one-step method, and that it involves the evaluation of f at an intermediate value of t, namely $t_n+h/2$.

In order to conform to a standard notation to be used later, we rewrite this algorithm as follows:

$$\begin{cases} k_0 = h f(y_n, t_n) \\ k_1 = h f(y_n + \frac{1}{2}k_0,\, t_n + \frac{1}{2}h) \\ y_{n+1} = y_n + k_1 \end{cases}$$

This method is an example of an explicit 2-stage Runge-Kutta method. The number of stages is the number of f's that appear. The method will turn out to be of order 2, just as the midpoint rule for quadrature is. However, this will not be proved until later.

Another well-known quadrature method is the <u>trapezoid rule</u>. Here the required area under $\dot{y}(t)$ is approximated by that of the trapezoid whose two heights are $\dot{y}(t_n)$ and $\dot{y}(t_{n+1})$ and whose width is h. Thus the associated ODE method would be

$$y_{n+1} = y_n + \frac{h}{2}(\dot{y}_n + \dot{y}_{n+1}).$$

This is an implicit method, as $\dot{y}_{n+1}=f(y(t_{n+1}),t_{n+1})$ is not explicitly available. However, as with the midpoint rule, we may choose to approximate required advanced values of y by use of the explicit Euler method:

$$y(t_{n+1}) \simeq y_n + h\dot{y}_n .$$

If so, we are led to the algorithm

$$\begin{cases} g_1 = y_n + h\dot{y}_n \\ y_{n+1} = y_n + \frac{h}{2}[f(g_1, t_{n+1}) + \dot{y}_n] \end{cases}.$$

We rewrite this in the standard notation as

$$\begin{cases} k_0 = hf(y_n, t_n) \\ k_1 = hf(y_n + k_0, t_n + h) \\ y_{n+1} = y_n + \frac{1}{2}k_0 + \frac{1}{2}k_1 \end{cases}.$$

This method does not involve intermediate t values, and so is not, strictly speaking, a true Runge-Kutta method. Despite this, it is regarded as a Runge-Kutta method (of a degenerate type perhaps), in that it fits into the same general formula pattern that defines the Runge-Kutta class, as will be seen shortly. (In the same vein, many one-step methods are included in the linear multistep class, even though they are not strictly multistep.)

The above is thus a second example of a 2-stage, 2^{nd} order, explicit Runge-Kutta method. The order of 2 arises from the 2^{nd} order accurate trapezoid rule on which it is based. The intermediate use of the Euler method of only first order does not, as it turns out, degrade the order of the overall method.

These two examples should be enough to suggest a general formula for explicit 2-stage Runge-Kutta methods, namely

$$\begin{cases} k_0 = h f(y_n, t_n) \\ k_1 = h f(y_n + \alpha k_0, t_n + \eta h) \\ y_{n+1} = y_n + \beta k_0 + \gamma k_1 \end{cases},$$

with some constant coefficients α, β, γ, and η. This defines an algorithm for ODE systems, but, depending on the coefficients, it is not automatically of order 2. However, we can determine what the order is, for a given set of coefficients. The calculations for this will be done fairly thoroughly for this case; they will be done much less thoroughly for later, more complicated, cases.

What has to be done to determine the order is to calculate terms in the Taylor series for y_{n+1} and $y(t_{n+1})$ under the assumption $y(t_n)=y_n$, and see how many of the terms agree. The difference between these two quantities is d_n, the local truncation error. In order to simplify the notation, we consider only the scalar case, and use subscripts on f to denote partial derivatives all of which will be evaluated at t_n. We first write the Taylor series for y_{n+1}, expanded about t_n and carried to h^3 terms:

$$k_0 = h\dot{y}_n$$

$$k_1 = h\left[f(y_n,t_n) + f_y \alpha k_0 + f_t \eta h + \tfrac{1}{2} f_{yy}(\alpha k_0)^2 + f_{yt}\alpha k_0 \eta h + \tfrac{1}{2} f_{tt}(\eta h)^2 + O(h^3)\right]$$

$$= h\dot{y}_n + h^2(\alpha f_y \dot{y}_n + \eta f_t) + h^3\left(\tfrac{1}{2}\alpha^2 f_{yy}\dot{y}_n^2 + \alpha\eta f_{yt}\dot{y}_n + \tfrac{1}{2}\eta^2 f_{tt}\right) + O(h^4)$$

$$y_{n+1} = y_n + h(\beta+\gamma)\dot{y}_n + h^2(\gamma\alpha f_y \dot{y}_n + \gamma\eta f_t) + h^3\left(\tfrac{1}{2}\gamma\alpha^2 f_{yy}\dot{y}_n^2 + \gamma\alpha\eta f_{yt}\dot{y}_n + \tfrac{1}{2}\gamma\eta^2 f_{tt}\right) + O(h^4).$$

Now we need the Taylor series for $y(t_{n+1})$, the true solution of the ODE for which $y(t_n)=y_n$. We have

$$y(t_n+h) = y_n + h\dot{y}_n + \tfrac{1}{2}h^2\ddot{y}_n + \tfrac{1}{6}h^3\dddot{y}_n + O(h^4),$$

but must express this in terms only of f and its partial derivatives, in order to make the comparison. Thus we write

$$\dot{y} = f$$
$$\ddot{y} = f_y \dot{y} + f_t$$
$$\dddot{y} = f_y \ddot{y} + f_{yy} \dot{y}^2 + 2 f_{yt} \dot{y} + f_{tt} .$$

Now we can compare the two series, and recall that as the coefficients of terms up to h^k agree, the order is at least k. The conditions for matching of the successive terms are as follows:

h^0 : (match)
h^1 : $\beta + \gamma = 1$
h^2 : $\gamma\alpha = \frac{1}{2}$, $\gamma\eta = \frac{1}{2}$
h^3 : $\frac{1}{2}\gamma\alpha^2 = \frac{1}{6}$, $\gamma\alpha\eta = \frac{1}{3}$, $\frac{1}{2}\gamma\eta^2 = \frac{1}{6}$, $0 = \frac{1}{6}$.

Notice that a match of the two terms in h^k requires that the individual terms within each coefficient expression must match, in order to assure equality of the coefficients for <u>general</u> f. One could also show that the individual terms must match by choosing various special cases for f so as to get the individual equations above one at a time. In particular the matching of the $f_y \ddot{y}$ terms shows that the h^3 terms can not generally agree, regardless of the values of the constants. Only in the special case that $f_y \ddot{y} = 0$ is such a match possible. This means that we can never achieve order 3 or more for this class of methods.

The conclusions to be drawn as to order are therefore as follows: We get order 1 or more if $\beta = 1 - \gamma$. This is thus the condition of

consistency, or the condition that the method is exact if f is a constant. We get order 2 for the method if, in addition to the consistency condition, we have $\alpha = \eta = 1/2\gamma$. No higher order is possible.

This doesn't say what γ should be, except that it cannot vanish. In fact, for any choice of $\gamma \neq 0$, with α, β, and η as given above, we get a valid 2^{nd} order method. In other words, we have a one-parameter family of 2^{nd} order, explicit, 2-stage Runge-Kutta methods;

$$\begin{cases} k_0 = h f(y_n, t_n) \\ k_1 = h f\left(y_n + \frac{1}{2\gamma} k_0,\ t_n + \frac{1}{2\gamma} h\right) \\ y_{n+1} = y_n + (1-\gamma) k_0 + \gamma k_1 \ . \end{cases}$$

The choice $\gamma = 1$ gives the midpoint rule, and $\gamma = \frac{1}{2}$ gives modified trapezoid rule. This completes the proof that these two examples given earlier are indeed of order 2.

The above analysis provides one further piece of information. It gives the local error term, as approximated by the difference between the h^3 terms of the two series. That is, assuming that the order 2 conditions hold, we have a local truncation error of

$$d_n = y_{n+1} - y(t_{n+1}) = h^3 \varphi(t_n) + O(h^4),$$

$$\varphi = \left(\tfrac{1}{2}\gamma\alpha^2 - \tfrac{1}{6}\right) f_{yy}\dot{y}^2 + \left(\gamma\alpha\eta - \tfrac{1}{3}\right) f_{yt}\dot{y}$$

$$+ \left(\tfrac{1}{2}\gamma\eta^2 - \tfrac{1}{6}\right) f_{tt} - \tfrac{1}{6} f_y \ddot{y}$$

$$= \left(\frac{1}{8\gamma} - \frac{1}{6}\right)\left(f_{yy}\dot{y}^2 + 2 f_{yt}\dot{y} + f_{tt}\right) - \frac{1}{6} f_y \ddot{y}.$$

This formula shows that the choice $\gamma=3/4$ gives the much neater formula $\varphi = -\frac{1}{6} f_y \ddot{y}$. However, reducing the number of terms in φ does not necessarily mean a reduction in the size of φ.

Note that, even in the special case $\gamma = 3/4$, and certainly in the general case, φ is not readily estimated. That is, d_n is not approximated by some expression such as $Ch^3\dddot{y}$ which can be easily estimated in practice, as is the case for the linear multistep methods. This is a major drawback of the Runge-Kutta methods, one which much attention has been paid to in recent research. There has been some success in removing this difficulty, which will be mentioned later (Sections (c) and (f)).

8. (b) Explicit r-stage; classical case

We can clearly generalize the process of the previous section. Instead of just one auxiliary intermediate point $(y_n+\alpha k_0, t_n+\alpha h)$, we can consider several such points. The number of these points, plus 1, is the number of stages (the number of values of f).

The case of three stages would have the following general form:

$$\begin{cases} k_0 = h f(y_n, t_n) \\ k_1 = h f(y_n + \beta_{11} k_0, t_n + \beta_{11} h) \\ k_2 = h f(y_n + \beta_{21} k_0 + \beta_{22} k_1, t_n + (\beta_{21} + \beta_{22}) h) \\ y_{n+1} = y_n + \gamma_0 k_0 + \gamma_1 k_1 + \gamma_2 k_2 . \end{cases}$$

Notice that this algorithm uses a coefficient of h in the t argument which is the sum of the coefficients of the k_j in the y argument in each f expression. This means that in the simple case f=constant, the t argument has the correct value in correspondence to the value of the y argument. It turns out that if a more general algorithm, with this condition not imposed, were studied, the order conditions would result in that condition anyway. This was the case with two stages, where $\alpha = \eta$.

To simplify the notation slightly, we set

$$\alpha_1 = \beta_{11}, \quad \alpha_2 = \beta_{21} + \beta_{22}.$$

As shown in the text* we can calculate and compare the Taylor series for y_{n+1} and $y(t_{n+1})$ in this case, just as was done before for two stages, but with greater complexity involved. If we do this through the h^3 terms, we arrive at

$$h^1: \quad \gamma_0 + \gamma_1 + \gamma_2 = 1 \qquad \text{(consistency)}$$
$$h^2: \quad \gamma_1 \alpha_1 + \gamma_2 \alpha_2 = 1/2$$
$$h^3: \quad \tfrac{1}{2}\gamma_1 \alpha_1^2 + \tfrac{1}{2}\gamma_2 \alpha_2^2 = \tfrac{1}{6}, \quad \gamma_2 \alpha_1 \beta_{22} = 1/6.$$

*The text uses a special notation, which makes the method slightly easier to write down, and much easier to analyze. It involves defining, for the scalar case, vectors

$$\underset{\sim}{y} = \begin{pmatrix} t \\ y \end{pmatrix}, \quad \underset{\sim}{f} = \begin{pmatrix} 1 \\ f \end{pmatrix} = \begin{pmatrix} f^0 \\ f^1 \end{pmatrix}, \quad \underset{\sim}{k_\beta} = \begin{pmatrix} h \\ hf(\cdots) \end{pmatrix} = \begin{pmatrix} k_\beta^0 \\ k_\beta^1 \end{pmatrix}.$$

Thus for example $\underset{\sim}{k}_1 = h\underset{\sim}{f}(\underset{\sim}{y}_n + \alpha_1 \underset{\sim}{k}_0)$ is to be read as $k_1^1 = hf(y_n + \alpha_1 k_0^1, t_n + \alpha_1 h)$. The use of subscript indices (0 or 1) further simplifies the expressions for derivatives of y(t). This notation, while convenient for research in Runge-Kutta methods, is not necessary here.

This is a set of 4 equations in 6 unknown constants. Hence we would expect to get a solution with two degrees of freedom. This can be done, with α_1 and α_2 as the two free parameters, with the result

$$\gamma_1 = \frac{3\alpha_2 - 2}{6\alpha_1(\alpha_2 - \alpha_1)}, \quad \gamma_2 = \frac{3\alpha_1 - 2}{6\alpha_2(\alpha_1 - \alpha_2)},$$

$$\gamma_0 = 1 - \gamma_1 - \gamma_2,$$

$$\beta_{21} = \alpha_2 - 1/6\alpha_1\gamma_2,$$

$$\beta_{11} = \alpha_1, \quad \beta_{22} = \alpha_2 - \beta_{21},$$

$$(\alpha_1 \neq \alpha_2, \ \alpha_i \neq 0, \ \alpha_1 \neq 2/3).$$

For coefficients given by these relations, the algorithm is a 3rd order, explicit, 3-stage Runge-Kutta method. It has a local error

$$d_n = h^4 \varphi(t_n) + O(h^5),$$

with φ being a rather complicated combination of f and its derivatives. Again, there is no choice of the parameters which will make $\varphi \equiv 0$, and so 3 is the maximal order here.

Now we can write the full generalization (still explicit) to an r-stage method. Its general form is

$$k_0 = h f(y_n, t_n),$$

$$k_\ell = h f\left(y_n + \sum_{j=1}^{\ell} \beta_{\ell j} k_{j-1}, \ t_n + \alpha_\ell h\right),$$

$$\ell = 1, 2, \cdots, r-1, \qquad \alpha_\ell = \sum_{j=1}^{\ell} \beta_{\ell j},$$

$$y_{n+1} = y_n + \sum_{\ell=0}^{r-1} \gamma_\ell k_\ell .$$

This algorithm can be characterized in a more compact form simply by the triangular matrix $(\beta_{qj})_{1 \le q \le r, 1 \le j \le q}$ and the vector $(\gamma_q)_{0 \le q \le r-1}$. This is a set of $r + r(r-1)/2 = r(r+1)/2$ coefficients. The various order conditions are, in principle, obtainable with Taylor series as before, but are so complex that they cannot even be written down in complete generality.

For the case r=4, the conditions for order 4 are given in the text (p.35). They constitute 8 relations in 10 unknowns, and they have a 2-parameter family of solutions, in terms of α_1 and α_2, with $\alpha_1 \neq \alpha_2$, $\alpha_1 \neq 1$, $\alpha_2 \neq 1$, $\alpha_1 \neq 0 \neq \alpha_2$. The result that the h^{r+1} term in d_n cannot be made to vanish identically, as seen for r=2 and r=3, holds for r=4 as well, and in fact for any r (see Section (c)).

One of the solutions to the 8 relations for order 4 is the classical Runge-Kutta method, given by

$$(\beta_{ij}) = \begin{pmatrix} \beta_{11} & 0 & 0 \\ \beta_{21} & \beta_{22} & 0 \\ \beta_{31} & \beta_{32} & \beta_{33} \end{pmatrix} = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$(\gamma_q)_0^3 = (1/6, 1/3, 1/3, 1/6), \quad \alpha_1 = \alpha_2 = \tfrac{1}{2}, \quad \alpha_3 = 1.$$

(The algorithm in the usual form was given in Section 5(c).) Notice that the above values of α_1 and α_2 cannot be substituted into the 2-parameter solution equations, because of the denominators of $\alpha_1 - \alpha_2$. But if $\alpha_1 = \frac{1}{2} + \delta$, and $\alpha_2 = \frac{1}{2} - \delta$ are inserted, the result simplified, and δ is made to go to zero, the above coefficient values do result.

The classical Runge-Kutta method, like previous examples, arises from a quadrature rule, namely Simpson's rule:

$$\Delta y_n = \int_{t_n}^{t_n+h} \dot{y}(t)\,dt \approx \frac{h}{6}\left[\dot{y}(t_n) + 4\,\dot{y}(t_n+\tfrac{h}{2}) + \dot{y}(y_n+h)\right].$$

The identification arises by making the approximations

$$\frac{2}{3}h\dot{y}(t_n+\tfrac{h}{2}) \approx \tfrac{1}{3}k_1 + \tfrac{1}{3}k_2, \quad \frac{h}{6}\dot{y}(t_n+h) \approx \tfrac{1}{6}k_3,$$

although this choice of strategy for the Runge-Kutta algorithm is not obvious from Simpson's rule alone. In fact, Runge's original method of 1895 was not that given above. It was Kutta in 1901 who revised the fomulas by using undetermined coefficients, chosen to obtain maximum order.

The question of efficiency is relatively easy to look at for the classical Runge-Kutta method. As a 4-stage explicit method, it requires 4 f evaluations to do each step. However, in order to control the error, the classical method of step doubling (or halving) is usually used (see Section 7 (c)). In that case, another 4 evaluations are made for a second step size h, and then a single step is taken over the interval of size 2h. For the latter, instead of 4 evaluations of f, only 3 are needed, because the first one, $f(y_n,t_n)$ has already been done. So a total of 4+4+3=11 evaluations is used to integrate from t_n to t_n+2h and generate an error estimate at the same time. In terms of steps of size h, the cost is therefore $5\frac{1}{2}$ evaluations per step. This number will be recalled later for comparison purposes.

8. (c) Order, error estimation

The examples of r-stage explicit Runge-Kutta methods given so far indicate that the order can never be made larger than r. This fact can be proved for any r.

Theorem: For an r-stage explicit Runge-Kutta method, the order is $\leq r$.

Proof: For the purpose of this proof, it is sufficient to consider simply the scalar ODE $\dot{y} = \lambda y$, where λ is an arbitrary (say, real) constant. The method then gives (letting $x = h\lambda$)

$$k_0 = h f(y_n, t_n) = h\lambda y_n = x\, y_n ,$$

$$k_q = h f\left(y_n + \sum_1^q \beta_{qj} k_{j-1},\ t_n + \alpha_q h\right)$$

$$= x\left(y_n + \sum_1^q \beta_{qj} k_{j-1}\right), \quad q = 1, 2, \cdots, r-1 .$$

Examining the dependence of k_q on x, we find that k_q is a polynomial in x of degree q+1, multiplied by y_n. This is clearly true for k_0, and if it is true for $k_0, k_1, \cdots, k_{q-1}$, then the formula for k_q shows that it is true for k_q. It follows that $y_{n+1} = \sum_0^{r-1} \gamma_q k_q$ is a polynomial in x of degree r, times y_n. But for this problem, the true solution with $y(t_n) = y_n$ satisfies

$$y(t_{n+1}) = y(t_n + h) = e^{h\lambda}\, y(t_n) = e^x\, y_n$$

$$= \sum_{q=0}^{\infty} (x^q / q!)\, y_n .$$

Thus $d_n = y_{n+1} - y(t_{n+1})$ is a series in x in which at most the terms through $x^r/r!$ cancel. That is d_n has an x^{r+1} term, and possibly lower order terms

as well. But an order of r+1 or more would require $d_n = O(h^{r+2})$, which is the same as $d_n = O(x^{r+2})$. This is a contradiction. QED

As the various explicit r-stage Runge-Kutta methods are studied more closely for r=5,6,---, it turns out they lead to more equations to be satisfied for order r than there are coefficients. Hence the 5-stage methods only have order 4 at most, etc. This discrepancy gets worse as r increases. (See the table in the text, p. 37.)

As indicated earlier, there is difficulty in estimating local error for a given Runge-Kutta method. In many cases, the easiest and most reliable way to do this is the doubling/halving technique described in Section 7 (c). However, because this increases the cost per step from 4 to (effectively) $5\frac{1}{2}$ evaluations of f, other approaches have been taken instead.

A choice proposed by Merson, which contains an error estimate, is the 5-stage, 4^{th} order, explicit Runge-Kutta given by

$$(\beta_{8j}) = \begin{pmatrix} 1/3 & 0 & 0 & 0 \\ 1/6 & 1/6 & 0 & 0 \\ 1/8 & 0 & 3/8 & 0 \\ 1/2 & 0 & -3/2 & 2 \end{pmatrix}, \quad (\gamma_8) = \begin{pmatrix} 1/6 \\ 0 \\ 0 \\ 2/3 \\ 1/6 \end{pmatrix}.$$

It uses also the intermediate quantity

$$\tilde{y}_{n+1} = y_n + \frac{1}{2}k_0 - \frac{3}{2}k_2 + 2k_3 ,$$

which is the y argument of k_4. (See p. 85 in the text.) It can be shown that if f is linear in y and t, then the local error is

$$d_n = -\frac{1}{5}(y_{n+1} - \tilde{y}_{n+1}) + O(h^6),$$

and hence is easily estimated. However, to the extent that f is non-linear, that estimate will be inaccurate, although the local error d_n remains $O(h^5)$. Note that only 5 evaluations per step are required by this method.

A third approach used to estimate error in Runge-Kutta methods is the idea of imbedding. Here an r-stage method of order p is imbedded in an (r+1)-stage of order p+1, in the sense that all the f evaluations required by the first method are also required by the second. Then the difference between the two resulting approximations to $y(t_{n+1})$ is a good estimate, to within $O(h^{p+1})$, of the error in the p^{th} order result. Names associated with imbedding methods are Fehlberg and Sarafyan. The Fehlberg methods will be presented in Section (f).

8. (d) Implicit r-stage

If we consider the explicit Runge-Kutta methods, and drop the requirement that each k_q depend explicitly on the previous k's only, we get the class of implicit Runge-Kutta methods. That is each k_q is defined by a relation that may involve all the k's, and so is defined

implicitly only. With a shift in notation from the previous sections, we write the general r-stage implicit Runge-Kutta method as follows:

$$\begin{cases} k_1 = h f(y_n + \beta_{11} k_1 + \beta_{12} k_2 + \cdots + \beta_{1r} k_r,\ t_n + \alpha_1 h) \\ k_2 = h f(y_n + \beta_{21} k_1 + \quad \cdots \quad + \beta_{2r} k_r,\ t_n + \alpha_2 h) \\ \cdots\cdots \\ k_r = h f(y_n + \beta_{r1} k_1 + \quad \cdots \quad + \beta_{rr} k_r,\ t_n + \alpha_r h) \\ \qquad (\alpha_q = \beta_{q1} + \beta_{q2} + \cdots + \beta_{qr}) \\ y_{n+1} = y_n + \gamma_1 k_1 + \cdots + \gamma_r k_r \ . \end{cases}$$

If we keep this standard structure in mind, then it suffices to specify a matrix and a vector:

$$\beta = (\beta_{ij})_{i,j=1}^{r} = \begin{pmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1r} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2r} \\ \cdots & & & \\ \beta_{r1} & \cdots & & \beta_{rr} \end{pmatrix},$$

$$\gamma = (\gamma_i)_1^r = (\gamma_1, \gamma_2, \cdots, \gamma_r) \ .$$

This same notation can be used for the explicit case also, so that the above system actually includes both types of Runge-Kutta methods. In the explicit case, the matrix β is strictly lower triangular: only the β_{ij} with $i > j$ can be nonzero (the others are 0).

The meaning of a stage changes slightly for the implicit case. The number of stages is the number of f expressions that appear in the defining relations. It is not necessarily the same as the number of f evaluations that are necessary to implement the method.

A simple example of a 2-stage implicit Runge-Kutta method is the following:

$$\begin{cases} k_1 = h f(y_n, t_n) \\ k_2 = h f(y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2, t_n + h) \\ y_{n+1} = y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 \ . \end{cases}$$

On close examination this is recognizable as the trapezoid rule, since $k_1 = h\dot{y}_n, k_2 = h\dot{y}_{n+1}$. As such it is also a linear multistep method and is usually thought of in that class, not the Runge-Kutta class.

If the matrix β is more or less full, the method is called <u>fully implicit</u>. Methods of this type are due mainly to J.C. Butcher. In order to implement such a method, it is necessary to solve a system of rN (generally nonlinear) algebraic equations, since each of r quantities k_q is a vector of length N. This is a difficult job for only N equations, and is even harder for rN, if we consider r up to 5 or so.

A compromise is to make β not so full, but only lower triangular:

$$\beta = \begin{pmatrix} \beta_{11} & & 0 \\ \beta_{21} & \beta_{22} & \\ \cdots & & \\ \beta_{r1} & \cdots & \beta_{rr} \end{pmatrix}.$$

This means we are reducing the implicitness of the relations to only one k_q at a time:

$$k_q = h f(y_n + \beta_{q1} k_1 + \cdots + \beta_{qq} k_q, t_n + \alpha_q h).$$

This is only N equations to solve, but it must be done r times to get all of the k_q. These methods are called semi-implicit Runge-Kutta methods, and are due mainly to H.H. Rosenbrock.

In either case, the discussion in Section 7 (b) applies here, in the actual solution of the implicit equations. The easiest method of solution is functional iteration, where only repeated evaluations of f are called for. In the semi-implicit case, a quasi-Newton iteration method could be used, but the matrix manipulation that is necessary there becomes impractical in the fully implicit case, in general. For some special problems, however, the use of implicit methods, with specialized techniques for solving the implicit relations, may be the best choice.

The calculation of order for the implicit Runge-Kutta methods is more complicated than in the explicit case. One generates Taylor series as before, but one has to insert these series repeatedly into computed series in order to get enough terms explicitly, i.e. to eliminate the implicitness. This is done for the 2-stage case in the text (pp 37-38). For the general r-stage case, there are r^2+r free parameters, and it has been shown that this number is more than enough to satisfy the conditions for order r. E.g., the conditions for order 3 for the 2-stage method yield 4 equations in 6 unknown coefficients, and hence a 2-parameter

family of solutions, each giving a 3^{rd} order method. In fact, Butcher showed that there are enough free parameters to satisfy the conditions for order 2r for an r-stage method. The text gives examples for r=2 of a 3^{rd} order and a 4^{th} order method, both fully implicit.

8. (e) Higher-order methods; Fehlberg's methods; numerical examples*

Numerical procedures for solving ordinary differential equations with given initial conditions occupy an important sector of scientific research. Runge-Kutta types are a highly regarded subset of these numerical techniques, and subsets of Runge-Kutta types proliferate current mathematical literature. An introduction to various aspects of general explicit Runge-Kutta methods, and Fehlberg's mutation in particular, follow.

General Approach

Given: A first order differential equation

$$\frac{dy}{dx} = f(x,y)$$

with an initial condition

$$y(x_0) = y_0.$$

---

*This section of the notes and course was provided by R.L. Pexton. While the notation used differs slightly from that of the preceding sections, this is a completely self-contained study of explicit Runge-Kutta methods.

The "true" solution at the point $x_0 + h$, where h is a real number (positive or negative), may be closely approximated by a Taylor series expansion

$$y(x_0+h) \simeq y(x_0) + \frac{h}{1!}\, y'(x_0)\, \frac{h^2}{2!}\, y''(x_0) + \frac{h^3}{3!}\, y'''(x_0) + \frac{h^4}{4!}\, y^{iv}(x_0) + \ldots.$$

If $f(x,y)$ is <u>easily</u> differentiated, this is a reasonable method to use. However, in general, the higher order derivatives are complicated--both difficult to derive and to evaluate.

To avoid these possible difficulties, Runge-Kutta methods approximate the "true" solution at $x_0 + h$ by summing a linear combination of particular values of $f(x,y)$. The evaluation points are selected such that if these particular f's are expanded in a Taylor series about (x_0,y_0), the summed series will agree exactly, to some order of h, with the Taylor series expansion of the "true" solution.

$$y(x_0+h) \simeq y(x_0) + h[c_0 f(x_0,y_0) + c_1 f(x_0+a_1 h, y_0+b_1 h) + c_2 f(x_0+a_2 h, y_0+b_2 h) + \ldots].$$

When the b's are expressed as a linear combination of the preceding values of f, the <u>usual</u> form of Runge-Kutta formulae are obtained:

$$f_0 = f(x_0,y_0)$$

$$f_1 = f(x_0+\alpha_1 h, y_0+\beta_{10} f_0)$$

$$f_2 = f(x_0+\alpha_2 h, y_0+\beta_{20} f_0+\beta_{21} f_1)$$

$$f_3 = f(x_0+\alpha_3 h, y_0+\beta_{30} f_0+\beta_{31} f_1+\beta_{32} f_2)$$

.

.

.

$$y(x_0+h) \simeq y(x_0) + h[c_0 f_0 + c_1 f_1 + c_2 f_2 + c_3 f_3 + \ldots]$$

Order of h Agreement

For Taylor series expansion of the "true" solution, use the given differential equation to develop the higher order derivatives.

$$y''(x) = \frac{d}{dx} y'(x) = \frac{d}{dx} f(x,y) = f_x + f_y f$$

$$y'''(x) = \frac{d}{dx} y''(x) = \frac{\partial}{\partial x} (f_x + f_y f) + \left[\frac{\partial}{\partial y} (f_x + f_y f)\right] f$$

$$= f_{xx} + 2f_{xy} f + f_{yy} f^2 + f_y (f_x + f_y f),$$

etc. To 3rd order we have

$$y(x_0+h) \approx y_0 + hf_0 + \frac{h^2}{2!} (f_x + f_y f_0) + \frac{h^3}{3!} \left[f_{xx} + 2f_{xy} f_0 + f_{yy} f_0^2 + f_y (f_x + f_y f_0)\right] + O(h^4)$$

An expansion, to 3rd order, of the f_k terms in the 4-stage formula in Taylor series is displayed in Table I. Notice that the coefficients of powers of h are similar in form to the binomial series expansion,

$$(a+b)^n = a^n + na^{n-1}b + \frac{n(n-1)}{2!} a^{n-2}b^2 + \frac{n(n-1)(n-2)}{3!} a^{n-3}b^3 + \ldots .$$

Equations of Condition

Equate like powers of h in the two Taylor series expressions. We restrict the Runge-Kutta expansion to 3rd order. The results are shown on p. 97, following Table I.

Table I

$$y(x+h) = y + h\left(c_0 f_0 + c_1 f_1 + c_2 f_2 + c_3 f_3\right) + O(h^4)$$

$$f_1(x,y) = f(x + \alpha_1 h, y + h\beta_{10} f_0) = f_0 + h\left[\alpha_1 f_x + \beta_{10} f_0 f_y\right]$$
$$+ \frac{h^2}{2!}\left[\alpha_1^2 f_{xx} + 2\alpha_1\beta_{10} f_0 f_{xy} + (\beta_{10} f_0)^2 f_{yy}\right]$$
$$+ \frac{h^3}{3!}\left[\alpha_1^3 f_{xxx} + 3\alpha_1^2\beta_{10} f_0 f_{xxy} + 3\alpha_1(\beta_{10} f_0)^2 f_{xyy} + (\beta_{10} f_0)^3 f_{yyy}\right] + O(h^4)$$

$$f_2(x,y) = f\left(x + \alpha_2 h, y + h(\beta_{20} f_0 + \beta_{21} f_1)\right) = f_0 + h\left[\alpha_2 f_x + (\beta_{20} f_0 + \beta_{21} f_1) f_y\right]$$
$$+ \frac{h^2}{2!}\left[\alpha_2^2 f_{xx} + 2\alpha_2(\beta_{20} f_0 + \beta_{21} f_1) f_{xy} + (\beta_{20} f_0 + \beta_{21} f_1)^2 f_{yy}\right]$$
$$+ \frac{h^3}{3!}\left[\alpha_2^3 f_{xxx} + 3\alpha_2^2(\beta_{20} f_0 + \beta_{21} f_1) f_{xxy} + 3\alpha_2(\beta_{20} f_0 + \beta_{21} f_1)^2 f_{xyy} + (\beta_{20} f_0 + \beta_{21} f_1)^3 f_{yyy}\right] + O(h^4)$$

$$f_3(x,y) = f\left(x + \alpha_3 h, y + h(\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2)\right) = f_0 + h\left(\alpha_3 f_x + (\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2) f_y\right)$$
$$+ \frac{h^2}{2!}\left(\alpha_3^2 f_{xx} + 2\alpha_3(\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2) f_{xy} + (\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2)^2 f_{yy}\right)$$
$$+ \frac{h^3}{3!}\left(\alpha_3^3 f_{xxx} + 3\alpha_3^2(\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2) f_{xxy} + 3\alpha_3(\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2)^2 f_{xyy}\right.$$
$$\left. + (\beta_{30} f_0 + \beta_{31} f_1 + \beta_{32} f_2)^3 f_{yyy}\right) + O(h^4)$$

Taylor Series		Runge-Kutta
	1st Order Terms	
hf_0		$h[c_0f_0+c_1f_0+c_2f_0+c_3f_0]$
	$c_0+c_1+c_2+c_3 = 1$	
	2nd Order Terms	
$\frac{h^2}{2!}[f_x+f_yf_0]$		$h^2\{c_1[\alpha_1f_x+\beta_{10}f_0f_y] + c_2[\alpha_2f_x + (\beta_{20}f_0+\beta_{21}f_0)f_y]$ $+ c_3[\alpha_3f_x + (\beta_{30}f_0+\beta_{31}f_0+\beta_{32}f_0)f_y]\}$
f_x terms	$c_1\alpha_1+c_2\alpha_2+c_3\alpha_3 = 1/2$	
f_yf_0 terms	$c_1\beta_{10}+c_2(\beta_{20}+\beta_{21}) + c_3(\beta_{30}+\beta_{31}+\beta_{32}) = \frac{1}{2}$	
	$\sum_{\lambda=0}^{k-1}\beta_{k\lambda} = \alpha_k$	$k = 1,2,3$
	3rd Order Terms	
$\frac{h^3}{3!}[f_{xx}+2f_{xy}f_0+f_{yy}f_0^2]$		$\frac{h^3}{2!}\{c_1[\alpha_1^2f_{xx}+2\alpha_1\beta_{10}f_0f_{xy} + (\beta_{10}f_0)^2f_{yy}]$ $+ c_2[2\beta_{21}(\alpha_1f_x+\beta_{10}f_0f_y)f_y + \alpha_2^2f_{xx} + 2\alpha_2(\beta_{20}f_0+\beta_{21}f_0)f_{xy}$ $+ (\beta_{20}f_0+\beta_{21}f_0)^2f_{yy}]$ $+ c_3[2\beta_{31}[\alpha_1f_x+\beta_{10}f_0f_y]f_y + 2\beta_{32}[(\alpha_2f_x+\beta_{20}f_0+\beta_{21}f_0)f_y]f_y$ $+ \alpha_3^2f_{xx} + 2\alpha_3(\beta_{30}f_0+\beta_{31}f_0+\beta_{32}f_0)f_{xy}$ $+ (\beta_{30}f_0+\beta_{31}f_0+\beta_{32}f_0)^2f_{yy}]\}$
f_xf_y, $f_0f_y^2$ terms	$c_2\beta_{21}\alpha_1+c_3\beta_{31}\alpha_1+c_3\beta_{32}\alpha_2 = \frac{1}{6}$	
f_{xx}, f_{xy}, f_{yy} terms	$\frac{1}{2}(c_1\alpha_1^2+c_2\alpha_2^2+c_3\alpha_3^2) = \frac{1}{6}$	

The process of higher order terms, while straightforward, rapidly increases in algebraic manipulation complexity.

Fehlberg-Runge-Kutta

An interesting and valuable variation of Runge-Kutta methods has been developed by Erwin Fehlberg. In order to minimize truncation error and to reliably control an adjustable stepsize procedure, Fehlberg develops two Runge-Kutta formulas simultaneously.

Particular Case: 2nd/3rd Order

Given a first order differential equation $y' = f(x,y)$ and an initial condition $y(x_0) = y_0$, consider the following two equations:

$$y = y_0 + h \sum_{k=0}^{2} c_k f_k + O(h^3)$$

$$\hat{y} = y_0 + h \sum_{k=0}^{3} \hat{c}_k f_k + O(h^4)$$

where

$$f_0 = f(x_0, y_0),$$

$$f_k = f\left(x_0 + \alpha_k h,\ y_0 + h \sum_{\lambda=0}^{k-1} \beta_{k\lambda} f_\lambda\right) \qquad k = 1,2,3$$

$$\left[\alpha_k = \sum_{\lambda=0}^{k-1} \beta_{k\lambda}\right]$$

Note that f_0, f_1, and f_2 are the same in both equations. The "y" formula produces the calculated solution while the difference between "y" and "$\hat{y}$" provides the local truncation error.

As in the general case all f_k are expanded in a Taylor series. Equating coefficients of powers of h produces 8 equations in the unknowns c_k, $\hat{c}_k$, α_k, and $\beta_{k\lambda}$

$$c_0 + c_1 + c_2 = 1 \qquad \hat{c}_0 + \hat{c}_1 + \hat{c}_2 + \hat{c}_3 = 1$$

$$c_1\alpha_1 + c_2\alpha_2 = 1/2 \qquad \hat{c}_1\alpha_1 + \hat{c}_2\alpha_2 + \hat{c}_3\alpha_3 = 1/2$$

$$c_2\beta_{21}\alpha_1 - 1/6 = T_1 \qquad \hat{c}_2\beta_{21}\alpha_1 + \hat{c}_3(\beta_{31}\alpha_1 + \beta_{32}\alpha_2) = 1/6$$

$$1/2(c_1\alpha_1^2 + c_2\alpha_2^2) - 1/6 = T_2 \qquad 1/2(\hat{c}_1\alpha_1^2 + \hat{c}_2\alpha_2^2 + \hat{c}_3\alpha_3^2) = 1/6$$

where T_1 and T_2 are the local truncation errors of the "y" formula.

Fehlberg imposes the condition

$$f_3 = f(x+h,y)$$

Since

$$f_3 = f\Big(x + \alpha_3 h,\ y + h(\beta_{30}f_0 + \beta_{31}f_1 + \beta_{32}f_2)\Big)$$

$$y(x + h) = y(x) + h(c_0f_0 + c_1f_1 + c_2f_2),$$

we have

$$\alpha_3 = 1 \qquad \beta_{30} = c_0 \qquad \beta_{31} = c_1 \qquad \beta_{32} = c_2$$

Assuming

$$\hat{c}_1 = 0 \qquad \beta_{21}\alpha_1 = (1/2)\alpha_2^2 \qquad \beta_{31}\alpha_1 + \beta_{32}\alpha_2 = (1/2)\alpha_3^2$$

we may solve for $\hat{c}_2$ and $\hat{c}_3$ as functions of α_2.

$$\hat{c}_2 = 1/6\Big(1/\alpha_2(1 - \alpha_2)\Big)$$

$$\hat{c}_3 = 1/6\Big((2 - 3\alpha_2)/(1 - \alpha_2)\Big)$$

Designating $T_2 = -T_1$, one evaluates

$$c_1 = 1/3\Big((2 - 3\alpha_2)/\alpha_1(\alpha_1 - 2\alpha_2)\Big)$$

$$c_2 = 1/6\Big((3\alpha_1 - 4)/\alpha_2(\alpha_1 - 2\alpha_2)\Big)$$

$$T_1 = 1/12\ \alpha_1\Big((3\alpha_2 - 2)/(\alpha_1 - 2\alpha_2)\Big)$$

For $\alpha_2 = 2/3$ we see that $T_1 = T_2 = 0$. Unfortunately, this results in a degenerate set of equations. However, for values close to 2/3 the truncation factors will be small. Fehlberg selected $\alpha_2 = 27/40$ and $\alpha_1 = 1/4$.

Explicitly we have

$$y = y_0 + h(214/891f_0 + 1/33f_1 + 650/891f_2) + O(h^3)$$

$$\hat{y} = y_0 + h(533/2106f_0 + 800/1053f_2 - 1/78f_3) + O(h^4)$$

with a local truncation error of

$$TE = y-\hat{y} = (-23/1782f_0 + 1/33f_1 - 350/11583f_2 + 1/78f_3)h$$

Numerical Examples

Consider numerically solving the Airy differential equation

$$\frac{d^2y}{dx^2} - xy = 0.$$

Let $y' = z$, then $y'' = z' = xy$; i.e., we can express the Airy 2nd order differential equation as a coupled pair of 1st order differential equations

$$y' = z \qquad z' = xy.$$

In matrix-vector notation

$$y'(1) = y(2) \qquad y'(2) = xy(1)$$

For

$$Y = \begin{pmatrix} y(1) \\ y(2) \end{pmatrix} \qquad A = \begin{pmatrix} 0 & 1 \\ x & 0 \end{pmatrix}$$

we have

$$Y' = AY.$$

The following results were obtained on a CDC 6600 high speed computer. We employed a Fortran code based on Fehlberg's 5th order Runge-Kutta process and possessing a built-in stepsize control mechanism.

Case I. For x c (-5.5,0) we compute and record results at intervals of length = -.1. Since 2nd order differential equations have two independent solutions we chose as initial conditions (a) $y(0) = Ai(0)$, $Z(0) = \dot{Ai}(0)$, (b) $y(0) = Bi(0)$, $Z(0) = \dot{Bi}(0)$. (Ai and Bi are the symbols used to denote the two solutions of Airy's equation.) Program controls allowed a minimum stepsize of -.001 and the error (per interval step of -.1) was to be less than 10^{-6}. All recorded results were at least the required accuracy.

Case II. For x c (0,11) results at unit intervals were calculated. The error control was 10^{-8}, minimum stepsize = 10^{-3}, and initial conditions were $y(0) = Bi(0)$, $Z(0) = \dot{Bi}(0)$.

Case III. Conditions as in Case II except we require results at intervals of .2.

The requested accuracy was achieved in both Case II and III. In Case II the right-hand-side of the 1st order differential equations was evaluated 2296 times. For Case III only 2104 r-h-s evaluations were required, i.e., the "print-out" interval may reduce substantially the amount of machine work.

Case IV. Consider intial conditions

$$y(0) = Ai(0) \qquad Z(0) = \dot{Ai}(0)$$

and otherwise Case II controls. Given that Cases I, II, and III produce accurate results, what can one predict about Case IV?

Fehlberg's 5th order method calculated the values displayed in Table II. The values seem reasonable except perhaps for Ai(11). Moreover, the results are valid for x ε (0,5).

Knowing that $Ai(v) = 1/\pi\sqrt{v/3}\, K_{1/3}(w)$ where $w = 2/3v^{3/2}$ we may generate a series of check values. The Bessel function $K_{1/3}(w)$ can be evaluated very accurately. These highly reliable values reveal that the Fehlberg-Runge-Kutta solution is incorrect for all $x > 5$.

Table II.

x	Ai(x)
0	0.33502805
1	.13529
2	.034924
3	.006591
4	.0009516
5	.0001090
6	.000170
7	.0000879
8	.001302
9	.02331
10	.04947
11	12.3304

Actually the solution monotonically decreases as x increases. Let the reader beware of blindly accepting any computed results!

Tables III and IV dramatically illustrate the differences in work required by different orders of Fehlberg-Runge-Kutta schemes. For the two problems shown, the methods used were the following:

F-R-K-4-N-1 : 4^{th} order, first variation

F-R-K-4-N-2 : 4^{th} order, second variation

F-R-K-5-N : 5^{th} order

F-R-K-7-N : 7^{th} order

All these routines are based on Fehlberg's Runge-Kutta methods for a system of N ODE's. The tables show the number of evaluations of the right-hand side f(t,y) (cumulative), as of various t.

Table III.

Runge-Kutta Results

Fehlberg's approximations

R. England: Computer Journal, vol. 12, no. 2, p. 168.

$$\frac{d\gamma}{dt} = \left(\frac{V}{2\cdot 1E7+h} - \frac{32\cdot 2}{V}\right)\cos\gamma$$

$$\frac{dV}{dt} = \frac{32\cdot 2*1\cdot 25}{m} - 32\cdot 2\sin\gamma - \frac{5E\text{-}5\rho V^2}{m}$$

$$\frac{dm}{dt} = -\frac{1\cdot 25}{I}$$

$$\frac{dh}{dt} = V\sin\gamma$$

$$\frac{dx}{dt} = \frac{V\cos\gamma}{1+\frac{h}{2\cdot 1E7}}$$

$$\rho = 0\cdot 002378\exp\left(-\frac{h}{31000}\right)$$

$$I = 290 - 40\exp\left(-\frac{h}{27440 - 0\cdot 0771h}\right)$$

Initial conditions: $t = 0$, $\gamma = 1.569$, $V = 100$, $m = 1$, $h = x = 0$.

Print out at $t = 50, 100, 150, 200$.

For all orders

error = 10^{-8}, initial step = .5, minstep = .001 .

Step = 50

Method	Total Number of RHS Evaluations			
	t = 50	t = 100	t = 150	t = 200
F-R-K-4-N-1	1194	1410	1542	1728
F-R-K-4-N-2	1440	1686	1854	2106
F-R-K-5-N	672	792	872	984
F-R-K-7-N	234	299	338	416

Table IV.

Lapidus-Seinfeld: Numerical Solution of O.D.E, Example VIII, p. 84.

$$y_1' = \{-[40{\cdot}8 + 66{\cdot}7(M_1 + .08y_1)]y_1 + 66{\cdot}7(M_2 + .08y_2)y_2\}/z_1$$

$$y_i' = \{40{\cdot}8y_{i-1} - [40{\cdot}8 + 66{\cdot}7(M_i + .08y_i)]y_i + 66{\cdot}7(M_{i+1} + .08y_{i+1})y_{i+1}\}/z_i \qquad i = 2,3,4,5$$

$$y_6' = \{40{\cdot}8y_5 - [40{\cdot}8 + 66{\cdot}7(M_6 + .08y_6)]y_6\}/z_6$$

$$z_i = M_i + .16y_i + 75 \qquad i = 1(1)6$$

Initial conditions:

$$M = \begin{bmatrix} .73476500 \\ .74875687 \\ .75929635 \\ .76774008 \\ .77443837 \\ .77971110 \\ .78383672 \end{bmatrix} \qquad y(0) = \begin{bmatrix} -.03424992 \\ -.06192031 \\ -.08368619 \\ -.10042089 \\ -.11306320 \\ -.12243691 \end{bmatrix}$$

error = 10^{-8}, Step = 5, initial step = .5, minstep = .001 .

Method	Total Number of RHS Evaluations			
	t = 50	t = 100	t = 150	t = 200
F-R-K-4-N-1	792	1056	1296	1512
F-R-K-4-N-2	882	1188	1458	1728
F-R-K-5-N	504	720	896	1072
F-R-K-7-N	195	325	442	559

8. (f) Stability

The subjects of order and local error are considerations of accuracy. We now look at the question of stability for the Runge-Kutta methods.

We saw (Chapter 6) that a general one-step method is stable if the increment function ψ satisfies a Lipschitz condition. In the explicit case, it is clear that ψ will be Lipschitz as a result of f being Lipschitz, since ψ is explicitly constructed from values of f. In the implicit case, the same result holds, for all sufficiently small h, although it is not as obvious.

We turn then to the more specific question of absolute stability. Here we consider the simple test equation $\dot{y} = \lambda y$ (λ a complex constant), and we week the region S_a in the h plane where we will have $|y_{n+1}| \leq |y_n|$. Recall the reason for this consideration: If f is linear in y (or approximately so), and the eigenvalues of the coefficient matrix are λ_i, then all of the $h\lambda_i$ must lie in S_a if perturbations in y_n are to decay. If any are not, we can expect errors to grow unboundedly as $n \to \infty$ and thus destroy the accuracy of the computed answers. In this sense, stability is related to accuracy considerations.

We can easily calculate the results of a Runge-Kutta method on the problem $\dot{y} = \lambda y$. If we use the notation of Section (d), and insert into the general formula, we have, for q = 1,2,---,r,

$$k_q = h f(y_n + \sum_1^r \beta_{qj} k_j, t_n + \alpha_q h) = h\lambda (y_n + \Sigma \beta_{qj} k_j).$$

If the method is implicit, these relations are also. So it is easiest to break the discussion into two cases again.

Take first the explicit case, and consider the 2-stage, 2nd order methods of Section (a). For them, the above relations reduce to

$$k_1 = h\lambda y_n$$

$$k_2 = h\lambda (y_n + \frac{1}{2\gamma} k_1) = h\lambda (1 + \frac{h\lambda}{2\gamma}) y_n$$

$$y_{n+1} = y_n + (1-\gamma) k_1 + \gamma k_2$$

$$y_{n+1}/y_n = 1 + (1-\gamma) h\lambda + \gamma h\lambda (1 + h\lambda/2\gamma)$$

$$= 1 + h\lambda + (h\lambda)^2/2 .$$

Notice that these are just the first three terms of the Taylor series for $e^{h\lambda}$. In fact, we know in advance that y_{n+1}/y_n must agree with $e^{\lambda t_{n+1}}/e^{\lambda t_n} = e^{h\lambda}$ up to the h^2 term, because the method is of order 2. But for absolute stability, we want to know when $|y_{n+1}/y_n| \leq 1$. Thus for this case, the absolute stability region is

$$S_a = \{ h\lambda : |1 + h\lambda + (h\lambda)^2/2| \leq 1 \}$$

in the complex $h\lambda$ plane.

For the general explicit r-stage method, the relations defining k_q will give polyncmials in $h\lambda$, multiplying y_n. (See the theorem on order

in Section (c).) As a result y_{n+1}/y_n is a polynomial of degree r in $h\lambda$. The first few terms of that polynomial must be the terms $(h\lambda)^k/k!$ of the series for $e^{h\lambda}$, for the k up to the order of the method. Again, S_a is a region in the $h\lambda$ where that polynomial is at most 1 in magnitude. The region for the classical case is shown in the text (p. 41). One important feature of S_a in all these cases is that it is finite in extent. It cannot go indefinitely far in any direction, because of the way it is defined by a polynomial.

Now consider the implicit case. The example of the trapezoid rule has already been covered (Ex. 6.1), with the result

$$y_{n+1}/y_n = (1+h\lambda/2)/(1-h\lambda/2),$$
$$S_a = \{h\lambda : Re(h\lambda) \le 0\}.$$

Generally, for an implicit r-stage method, the equations for the k_q constitute an r x r linear system. The coefficients in that system will be either 1 or quantities $-h\lambda\beta_{qj}$. As a result, the determinant is a polynomial in $h\lambda$, say $q(h\lambda)$ of degree r or less, and the solution will give

$$y_{n+1}/y_n = p(h\lambda)/q(h\lambda),$$

for a polynomial p of degree r or less. Thus the absolute stability region is

$$S_a = \{h\lambda : |p(h\lambda)/q(h\lambda)| \le 1\}.$$

With appropriate choices of coefficients, S_a can be made to include the entire left half-plane, making the method A-stable. The selection of implicit Runge-Kutta methods is closely related to the subject of Padé approximations, which deals with approximations to e^x by rational functions $p(x)/q(x)$ (p and q being polynomials).

8. (g) Summary

It is difficult to generalize about ODE methods, especially for a class as large as the Runge-Kutta methods. Nevertheless the following remarks are offered as indications of the standing of the Runge-Kutta methods among ODE methods.

(1) They are one-step methods. This implies that
   (a) there is no startup problem, and discontinuities at the t_n do not cause difficulties, but
   (b) some of the efficiency of linear multistep methods is lost, in that past information is discarded.

(2) They are difficult to derive and analyze, but this should not be considered a detraction for the user of an existing method.

(3) The implicit methods are
   (a) relatively impractical, because of the large nonlinear systems, but
   (b) potentially A-stable, and therefore useful for stiff problems.

(4) The explicit methods
   (a) are easy to use, but
   (b) have a finite absolute stability region.

(5) The order of the method is fixed. Many problems are best solved with variable-order linear multistep methods.

(6) If fixed order is acceptable and the problem is not stiff, the newer Runge-Kutta methods (e.g. Fehlberg's) are good.

Exercise 8.1

Examine the pair of subroutines, DIFSUB and RK1 on pp 83-84 of the text. (Make the corrections given in the Errata, Appendix II. See also the listing following these exercises.) These perform the classicial Runge-Kutta method, one step at a time, with a variable step size based on a control of local error via halving. Show that RK1 in fact performs the classical 4^{th} order Runge-Kutta method. Specifically, identify, in terms of k_0, k_1, k_2, and k_3, the vectors Y2 and Y3 as of statement numbers 1, 2, and 3, and the vector Y1 of statement 4.

Exercise 8.2

Write a main program which reads N, t_o, t_{last}, h, and ϵ from a data card, and calls DIFSUB repeatedly to solve a given problem on $t_o \leq t \leq t_{last}$ with an initial step size h and a local error control constant ϵ.

Exercise 8.3

Use the package obtained above to solve the problem $\ddot{u} = -u$, $u_o=0$, $\dot{u}_o=1$, $0 \leq t \leq \pi/4$ (from Exercise 7.1). This requires

(a) statements in the main program initializing Y.

(b) output statements in the main program. At each step, output t, u, and the relative error in u.

(c) the subroutine DIFFUN (T,Y,DY) to compute DY = $\dot{y}$ given T = t and Y = y, i.e. to evaluate f(y,t).

(d) the data card. Use $\pi/4 = .785$, and $\epsilon = 10^{-8}$. Make a reasonable guess at h, using the formulas

$d_n \approx h^5 \varphi_n$ with $\varphi_n = - y_n^{(5)}/120$ here.

After running the problem, can you explain why the actual relative errors are so much less that ϵ ?

Exercise 8.4

Use the above Runge-Kutta package to solve the problem

$$\dot{y} = \lambda(y - e^{-t}) - e^{-t}, \quad y(0) = 1, \quad 0 \le t \le 10,$$

with $\epsilon = 10^{-5}$, for the two cases (a) $\lambda = -1$ and (b) $\lambda = -100$. To do this, appropriately modify the initialization of Y and the relative error calculation in the main program (use the fact that the solution is $y = e^{-t}$), the subroutine DIFFUN, and the data card (use a reasonable h). Also, in order to obtain control of relative error, set YMAX(1) = ABS(Y(1)) after each step in the main program. YMAX is what the estimated local error is compared to in the error control.

What is the average h for the two cases? Are they consistent with the S_a for the method?

```
      SUBROUTINE DIFSUB (N,T,Y,DY,H,HMIN,EPS,YMAX,ERROR,KFLAG,JSTART)
C*********************************************************************
C*    THE PARAMETERS TO THIS INTEGRATION SUBROUTINE HAVE              *
C*    THE FOLLOWING MEANINGS..                                        *
C*      N       THE NUMBER OF FIRST ORDER DIFFERENTIAL EQUATIONS      *
C*      T       THE INDEPENDENT VARIABLE                              *
C*      Y       THE DEPENDENT VARIABLES, UP TO 10 ARE ALLOWED.        *
C*      DY      AN ARRAY OF 10 LOCATIONS WHICH WILL CONTAIN THE       *
C*                VALUES OF THE DERIVATIVES AT THE START OF THE INTERVAL.*
C*      H       THE STEP SIZE THAT SHOULD BE ATTEMPTED.  IT MAY BE    *
C*                INCREASED OR DECREASED BY THE SUBROUTINE.           *
C*      HMIN    THE MINIMUM STEP SIZE THAT SHOULD BE ALLOWED ON THIS  *
C*                STEP.                                               *
C*      EPS     THE ERROR TEST CONSTANT.  THE ESTIMATED ERRORS ARE    *
C*                REQUIRED TO BE LESS THAN EPS*YMAX IN EACH COMPONENT.*
C*                IF YMAX IS ORIGINALLY SET TO +1 IN EACH COMPONENT,  *
C*                THE ERROR TEST WILL BE RELATIVE FOR THOSE COMPONENTS*
C*                GREATER THAN 1 AND ABSOLUTE FOR THE OTHERS.         *
C*      YMAX    THE MAXIMUM VALUES OF THE DEPENDENT VARIABLES ARE     *
C*                SAVED IN THIS ARRAY. IT SHOULD BE SET TO +1 BEFORE  *
C*                THE FIRST ENTRY. (SEE THE DESCRIPTION OF EPS).      *
C*      ERROR   THE ESTIMATED SINGLE STEP ERROR IN EACH COMPONENT     *
C*      KFLAG   A COMPLETION CODE WITH THE FOLLOWING MEANINGS..       *
C*                        -1    THE STEP WAS TAKEN WITH H = HMIN    *
C*                              BUT THE REQUESTED ERROR WAS NOT ACHIEVED.*
C*                        +1    THE STEP WAS SUCCESSFUL.            *
C*      JSTART AN INITIALIZATION INDICATOR WITH THE MEANING..       *
C*                        -1    REPEAT THE LAST STEP, RESTORING THE *
C*                              VALUES OF Y AND YMAX THAT WERE USED *
C*                              LAST TIME.                          *
C*                        +1    TAKE A NEW STEP.                    *
C*********************************************************************
      DIMENSION Y(10),DY(10),YMAX(10),YSAVE(10),Y1(10),Y2(10),Y3(10)
     1          ,ERROR(10),DYN(10),YMAXSV(10)
      IF(JSTART.LT.0) GO TO 2
C*********************************************************************
C* SAVE THE VALUES OF Y AND YMAX IN CASE A RESTART IS NECESSARY.     *
C*********************************************************************
      DO 1 I = 1,N
      YSAVE(I) = Y(I)
 1    YMAXSV(I) = YMAX(I)
C*********************************************************************
C*    CALCULATE THE INITIAL DERIVATIVES.                             *
C*********************************************************************
      CALL DIFFUN(T,Y,DYN)
      GO TO 4
C*********************************************************************
C*    RESTORE THE INITIAL VALUES OF Y AND YMAX FOR A RESTART.        *
C*********************************************************************
 2    DO 3 I = 1,N
      Y(I) = YSAVE(I)
 3    YMAX(I) = YMAXSV(I)
 4    KFLAG = 1
C*********************************************************************
C*    SAVE THE FINAL VALUE OF T AND CALCULATE THE HALF STEP.         *
C*********************************************************************
 5    A = H + T
      HHALF = H*0.5
C*********************************************************************
```

```
C*    PERFORM ONE FULL RUNGE KUTTA STEP                                *
C*******************************************************************
      CALL RK1(N,T,YSAVE,DYN,H,Y1)
C*******************************************************************
C*    NOW PERFORM TWO HALF INTERVAL RUNGE KUTTA STEPS                  *
C*******************************************************************
      CALL RK1(N,T,YSAVE,DYN,HHALF,Y2)
       THALF = T + HHALF
      CALL DIFFUN(THALF,Y2,DY)
      CALL RK1(N,THALF,Y2,DY,HHALF,Y3)
      ERRMAX = 0
C*******************************************************************
C*    CALCULATE THE NEW MAX Y2S, THE ERRORS AND THE MAX                *
C*    RELATIVE ERRORS.                                                 *
C*******************************************************************
      DO 6 I = 1,N
        YMAX(I) = AMAX1(YMAX(I),ABS(Y1(I)),ABS(Y2(I)),ABS(Y3(I)))
        ERROR(I) = ABS((Y3(I) - Y1(I))/15.0)
        ERRMAX = AMAX1(ERRMAX,ERROR(I)/(EPS*YMAX(I)))
C*******************************************************************
C*    CALCULATE THE IMPROVED VALUE OF Y BY ELIMINATING THE             *
C*    ESTIMATED ERROR.                                                 *
C*******************************************************************
        Y(I) = (16.0*Y3(I) -Y1(I))/15.0
 6      CONTINUE
      IF (ERRMAX.EQ.0) H = 2*H
      IF (ERRMAX.GT.0) H = H*ERRMAX**(-0.2)*0.99
      IF (ERRMAX.GT.1.0) GO TO 8
      KFLAG = 1
 7    T = A
      RETURN
 8    IF (H.GT.HMIN) GO TO 5
      IF (KFLAG.LT.0) GO TO 7
      H = HMIN
      KFLAG = -1
      GO TO 5
      END
```

```
      SUBROUTINE RK1(N,T,Y,DY,H,Y1)
C*************************************************************************
C*    THIS SUBROUTINE PERFORMS ONE RUNGE KUTTA STEP.                     *
C*      ARGUMENTS ARE..                                                  *
C*      N     -  NUMBER OF EQUATIONS.                                    *
C*      T     -  INITIAL VALUE OF INDEPENDENT VARIABLE.                  *
C*      Y     -  INITIAL VALUE OF DEPENDENT VARIABLES.                   *
C*      DY    -  INITIAL VALUE OF DERIVATIVES.                           *
C*      H     -  STEP SIZE                                               *
C*      Y1    -  THE ANSWER IS RETURNED HERE.                            *
C*************************************************************************
      DIMENSION Y(10),DY(10),Y1(10),Y2(10),Y3(10),DY1(10)
      HHALF = H*0.5
      DO 1 I = 1,N
    1   Y2(I) = Y(I) + HHALF*DY(I)
      CALL DIFFUN(T + HHALF,Y2,DY1)
      DO 2 I = 1,N
        Y3(I) = Y(I) + HHALF*DY1(I)
    2   Y2(I) = Y2(I) + 2*Y3(I)
      CALL DIFFUN(T + HHALF,Y3,DY1)
      DO 3 I = 1,N
        Y3(I) = Y(I) + H*DY1(I)
    3   Y2(I) = Y2(I) + Y3(I)
      CALL DIFFUN(T + H,Y3,DY1)
      DO 4 I = 1,N
    4   Y1(I) = (Y2(I) - Y(I) + HHALF*DY1(I))/3.0
      RETURN
      END
```

9. Linear Multistep Methods

This chapter deals with the large class of linear multistep methods, also referred to as finite difference methods. This subject was introduced earlier in Section 5 (d).

9. (a) Basic formulas, examples

The general formula for this class of ODE methods is

$$y_n = \sum_{i=1}^{K_1} \alpha_i y_{n-i} + h \sum_{i=0}^{K_2} \beta_i \dot{y}_{n-i} , \quad \dot{y}_j \equiv f(y_j, t_j).$$

Here we shall assume that the step size h is held fixed, so that the mesh points are $t_j = t_{j-1} + h$. The α_i and β_i are then fixed constants associated with the particular method. The numbers K_1 and K_2 determine how far back, in the sequences of y and $\dot{y}$ values, the method requires data. The number $K = \max(K_1, K_2)$ is the step number of the method, because there are K previous steps involved. The method is called a K-step method.

The above formula often appears in other forms, depending on various authors' notations. One such form is

$$\sum_{i=0}^{K_1} \alpha_i y_{n-i} + h \sum_{i=0}^{K_2} \beta_i \dot{y}_{n-i} = 0,$$

which can be obtained from the original by setting $\alpha_0 = -1$. However, in the latter form, one must add a normalization condition, such as $\alpha_0 = -1$, in order to specify the coefficients uniquely. Otherwise, multiplication of the equation by any nonzero constant gives a formula that defines the same method but has different coefficients. Another variant of the basic formula is

$$\sum_{i=0}^{K} (\alpha_i y_{n-i} + h \beta_i \dot{y}_{n-i}) = 0,$$

which can be obtained from the previous one by defining the last few α_i or β_i to be 0.

We have already seen three examples of linear multistep methods, although in fact they are only one-step methods. These are:

Explicit Euler: $K_1 = 1, \alpha_1 = 1, K_2 = 1, \beta_0 = 0, \beta_1 = 1;$

Implicit Euler: $K_1 = 1, \alpha_1 = 1, K_2 = 0, \beta_0 = 1;$

Trapezoid Rule: $K_1 = 1, \alpha_1 = 1, K_2 = 1, \beta_0 = \beta_1 = \frac{1}{2}.$

Let us now consider in detail a properly multistep example. The most general case in which $K_1=1$ and $K_2=2$ is given by

$$y_n = \alpha_1 y_{n-1} + h(\beta_0 \dot{y}_n + \beta_1 \dot{y}_{n-1} + \beta_2 \dot{y}_{n-2}).$$

This is a 2-step method. For a given set of coefficients (postponing for a moment what these should be), the first problem we face in implementing this method is that of starting it up. We would take our first step with n=2, and would require the data y_1, $\dot{y}_1$, and $\dot{y}_0$. The last item is known from the initial conditions, but the other two are not. What is generally done here is to use a one-step method, e.g. an explicit Runge-Kutta method, to take the step from t_0 to t_1. This produces y_1, and thus also $\dot{y}_1$, as required.

In performing the general step, we might think in terms of the history array we would use. The natural choice for this is the array, of size N x 3,

$$\underset{\sim}{y}_n = (y_n, \dot{y}_n, \dot{y}_{n-1}).$$

Then if $\underset{\sim}{y}_{n-1}$ is known, which it is when we have y_{n-1}, $\dot{y}_{n-1}$, and $\dot{y}_{n-2}$, the step consists of constructing y_n and then $\underset{\sim}{y}_n$ from it. In the explicit case, where $\beta_0=0$, this simply involves evaluating the formula for y_n, and filling in the rest of $\underset{\sim}{y}_n$ by use of f (for $\dot{y}_n$) and $\underset{\sim}{y}_{n-1}$ (for $\dot{y}_{n-1}$). In the implicit case, we would first have to solve the implicit equation for y_n, e.g. by one of the iterative methods discussed in Section 7 (b).

In the implicit case, for this or the more general linear multistep method, a convenient notation is often used for the necessary calculations in computing y_n. We use P to denote the operation of predicting y_n from the existing data -- i.e. from $\underset{\sim}{y}_{n-1}$ in this example. The result is the quantity $y_{n(o)}$ which is the first guess in the iterative process. We use E to denote an evaluation of f. The first such evaluation is generally that of $f(y_{n(o)}, t_n)$. We use C to denote the correcting of an iterate value $y_{n(m)}$ based on the previous evaluation. Thus, for example, an implementation might be briefly described simply as PEC, or PECE. If a number of iterations are done, say M of them, the shorthand description might be $P(EC)^M$ or $P(EC)^ME$.

Return now to the two-step formula, with four coefficients $\alpha_1, \beta_0, \beta_1$ and β_2, so far undetermined. We would naturally want to set these in a way that makes the formula as accurate as possible. Some ways to do this can be seen at once by choosing simple special cases for f. First, if $f \equiv 0$, the formula becomes $y_n = \alpha_1 y_{n-1}$, and this is clearly inaccurate unless $\alpha_1 = 1$, since the solution is y = constant. This condition will correspond to having an order of at least 0. Next, if $f \equiv$ constant, and

$\alpha_1=1$, the formula becomes $y_n=y_{n-1} + h(\beta_0+\beta_1+\beta_2)f$, which is accurate only if $\beta_0+\beta_1+\beta_2=1$, as the true solution satisfies $y(t_{n+1}) =y(t_n) + hf$. This condition corresponds to that of consistency, or of an order of at least 1. Thus we would impose these two conditions on the four parameters, resulting in a consistent method in which there are two degrees of freedom left.

The natural extension of this idea is to consider the cases of y quadratic in t (f linear), and then y cubic in t (f quadratic). If we take the case $y(t) = t^2$, $f = 2t$, the formula becomes

$$\begin{aligned} y_n &= t_{n-1}^2 + h(\beta_0 \cdot 2t_n + \beta_1 \cdot 2t_{n-1} + \beta_2 \cdot 2t_{n-2}) \\ &= t_{n-1}^2 + 2h[\beta_0(t_{n-1}+h) + \beta_1 t_{n-1} + \beta_2(t_{n-1}-h)] \\ &= t_{n-1}^2 + 2h(t_{n-1} + \beta_0 h - \beta_2 h). \end{aligned}$$

This is to be compared with

$$y(t_n) = t_n^2 = (t_{n-1}+h)^2 = t_{n-1}^2 + 2h\,t_{n-1} + h^2.$$

Thus we have an exact result in this case if and only if

$$\beta_0 - \beta_2 = 1/2.$$

This is the condition for the order to be at least 2. If we take the case $y(t) = t^3$, $f = 3t^2$, the formula becomes

$$\begin{aligned} y_n &= t_{n-1}^3 + h(\beta_0 \cdot 3t_n^2 + \beta_1 \cdot 3t_{n-1}^2 + \beta_2 \cdot 3t_{n-2}^2) \\ &= t_{n-1}^3 + 3h[\beta_0(t_{n-1}+h)^2 + \beta_1 t_{n-1}^2 + \beta_2(t_{n-1}-h)^2] \\ &= t_{n-1}^3 + 3h[(\beta_0+\beta_1+\beta_2)t_{n-1}^2 + 2(\beta_0-\beta_2)t_{n-1}h + (\beta_0+\beta_2)h^2] \\ &= t_{n-1}^3 + 3t_{n-1}^2 h + 3t_{n-1}h^2 + 3(\beta_0+\beta_2)h^3, \end{aligned}$$

assuming that the order 2 condition holds. When compared with

$$y(t_n) = (t_{n-1}+h)^3 = t_{n-1}^3 + 3t_{n-1}^2 h + 3t_{n-1}h^2 + h^3,$$

we see that the method gives the exact answer here if and only if

$$\beta_0 + \beta_2 = 1/3 .$$

This is the condition for the order to be at least 3. If we continue this process, we will obtain conditions which cannot be satisfied at the same time as those already obtained. Hence order 3, or exactness in the case $y(t)$ is a cubic in t, is the most we can get with this formula.

The calculation of these order conditions is much like that done earlier for the Runge-Kutta methods. However, there is one important difference: the formulas here are all linear in the coefficients. This makes the linear multistep methods somewhat easier to derive and analyze than the Runge-Kutta methods.

We can now write down the methods that arise in this example. First, if we want an explicit method, then the conditions for order 2, $\alpha_1=1$, $\beta_0+\beta_1+\beta_2=1$, $\beta_0-\beta_2=1/2$, together with the condition for explicitness, $\beta_0=0$, determine the coefficients uniquely: $\beta_2=-1/2$, $\beta_1=3/2$. This yields the formula

$$y_n = y_{n-1} + \frac{h}{2}(3\dot{y}_{n-1} - \dot{y}_{n-2}),$$

which is the explicit Adams, or Adams-Bashforth, method of order 2. If we make no restriction on β_0, but impose the condition for order 3, $\beta_0+\beta_2=1/3$, we obtain the unique solution $\beta_0 = 5/12$, $\beta_1 = 2/3$, $\beta_2 = -1/12$,

$$y_n = y_{n-1} + \frac{h}{12}(5\dot{y}_n + 8\dot{y}_{n-1} - \dot{y}_{n-2}).$$

This formula is the implicit Adams, or Adams-Moulton, method of order 3.

These formulas are often written in terms of backward differences. To do this, define

$$\nabla \dot{y}_n = \dot{y}_n - \dot{y}_{n-1}, \quad \nabla \dot{y}_{n-1} = \dot{y}_{n-1} - \dot{y}_{n-2}$$

$$\nabla^2 \dot{y}_n = \nabla \dot{y}_n - \nabla \dot{y}_{n-1} = \dot{y}_n - 2\dot{y}_{n-1} + \dot{y}_{n-2}.$$

Then the two Adams formulas above are

Explicit: $y_n = y_{n-1} + h\,(\dot{y}_{n-1} + \frac{1}{2}\nabla \dot{y}_{n-1})$

Implicit: $y_n = y_{n-1} + h(\dot{y}_n - \frac{1}{2}\nabla \dot{y}_n - \frac{1}{12}\nabla^2 \dot{y}_n)$.

Some of the other commonly used linear multistep methods are the following:

Milne: $y_n = y_{n-2} + \frac{h}{3}(\dot{y}_n + 4\dot{y}_{n-1} + \dot{y}_{n-2})$ (order 4)

Hamming: $y_n = \frac{9}{8} y_{n-1} - \frac{1}{8} y_{n-3}$

$$+\frac{3}{8}h(\dot{y}_n + 2\dot{y}_{n-1} - \dot{y}_{n-2}) \quad \text{(order 4)}$$

Gear: $y_n = \sum_{i=1}^{q} \alpha_i y_{n-i} + h\beta_0 \dot{y}_n$ (order q)

The method of Milne is obviously based directly on Simpson's rule for quadrature. All of these can be arrived at in the same manner as above.

9. (b) The Adams methods (explicit and implicit)

The group of methods referred to as the Adams methods include explicit and implicit methods of all orders. Adams and Bashforth presented the methods in 1883, and Moulton developed the implicit ones further in 1926.

The presentation here will be in terms of the backward difference (∇) notation, because it saves considerable writing. For any sequence $(z_1, z_2, \ldots)$, we define

$$\nabla^0 z_n = z_n$$
$$\nabla^1 z_n = \nabla z_n = z_n - z_{n-1}$$
$$\nabla^2 z_n = \nabla z_n - \nabla z_{n-1} = z_n - 2 z_{n-1} + z_{n-2}$$
$$\cdots$$
$$\nabla^{k+1} z_n = \nabla^k z_n - \nabla^k z_{n-1} .$$

It is clear that $\nabla^k z_n$ is a linear combination of $z_n, z_{n-1}, \ldots, z_{n-k}$, and in fact we can show (and will prove in Section (c)) that

$$\nabla^k z_n = \sum_{i=0}^{k} (-1)^i \binom{k}{i} z_{n-i} .$$

Much earlier in time, Newton gave formulas for interpolation polynomials, for use when a scalar function f(t) is known at equally spaced values t_k. If we write $f_k = f(t_k)$, $\nabla t_k = h$, and $s_k = t - t_k$, Newton's formulas can be written

$$f(t) = f_m + \frac{s_m \nabla f_m}{h} + \frac{s_m s_{m-1} \nabla^2 f_m}{2! h^2}$$
$$+ \cdots + \frac{s_m s_{m-1} \cdots s_{m-k+2} \nabla^{k-1} f_m}{(k-1)! h^{k-1}}$$
$$+ s_m s_{m-1} \cdots s_{m-k+1} f^{(k)}(\xi)/k! ,$$

where ξ is some point in the interval containing t, t_m, and t_{m-k+1}. If the last term is neglected, this means we are interpolating by a polynomial in t to approximate f at t from f at $t_m, t_{m-1}, \ldots, t_{m-k+1}$.

Let $f(t)= \dot{y}(t)$ and integrate that formula with respect to t over (t_{n-1},t_n). The result is an approximation of $y(t_n)-y(t_{n-1}) = \nabla y(t_n)$ as a linear combination of $f_m, \ldots, f_{m-k+1}$. The coefficients are integrals over (t_{n-1},t_n) of the coefficients that appear above, which are polynomials in t.

For the explicit methods, we take m=n-1. The result is the formula

$$\nabla y_n = y_n - y_{n-1} = h\dot{y}_{n-1} + h\gamma_1 \nabla \dot{y}_{n-1} + \cdots + h\gamma_{k-1}\nabla^{k-1}\dot{y}_{n-1}$$

with constants γ_j defined by

$$\int_{t_{n-1}}^{t_n} s_m s_{m-1} \cdots s_{m-j+1}\, dt = j!\, h^{j+1}\gamma_j ,$$

or

$$\gamma_j = (-1)^j \int_0^1 \frac{(-s)(-s-1)\cdots(-s-j+1)}{j!}\, ds$$

$$= (-1)^j \int_0^1 \binom{-s}{j}\, ds \qquad (\gamma_0 = 1).$$

In the case of a scalar problem (N=1) we can integrate also the error term, and get a formula for the error in y_n, assuming all the past data $(y_{n-1}, \dot{y}_{n-1}, \ldots, \dot{y}_{n-k})$ is exact. The result is

$$d_n \equiv y(t_n) - y_n = \gamma_k h^{k+1} y^{(k+1)}(\xi), \quad \xi \in (t_{n-k}, t_n).$$

For any N, we have

$$d_n = \gamma_k h^{k+1} y^{(k+1)}(t_n) + O(h^{k+2}).$$

To summarize, we have derived the formula

$$y_n = y_{n-1} + h\sum_{i=0}^{k-1} \gamma_i \nabla^i \dot{y}_{n-1}.$$

This is the explicit Adams or Adams-Bashforth formula. Notice that it is of order k, in that the local truncation error is $O(h^{k+1})$; that it is

a k-step method, in that it involves $\dot{y}_{n-k}$ but no earlier data; and that it is explicit, in that $\dot{y}_n$ does not appear. It can be written in the standard form

$$y_n = y_{n-1} + h \sum_{i=1}^{k} \beta_i \dot{y}_{n-i} \qquad (K_1 = 1,\ K_2 = k),$$

but then the β_i depend on k, while the γ_i do not. Thus for practical reasons, these methods are usually implemented in backward difference form, especially when variation of the order is to be allowed. The use of backward differences also allows for an easy estimation of the local error: Since $h^{k+1} y^{(k+1)}$ is approximately $h\nabla^k \dot{y}$ (within $O(h^{k+2})$), we have

$$d_n \approx \gamma_k h \nabla^k \dot{y}_{n-1}.$$

This is to be expected, because the first neglected term in the k^{th} order formula, compared to higher order formulas, is $\gamma_k h \nabla^k \dot{y}_{n-1}$.

To get the implicit formulas, return to the Newton interpolation formula for f(t), and let m=n. Then integration over (t_{n-1}, t_n) gives (neglecting the last term)

$$\nabla y(t_n) \approx h \dot{y}_n + h \gamma_1^* \nabla \dot{y}_n + \cdots + h \gamma_{k-1}^* \nabla^{k-1} \dot{y}_n,$$

where

$$\gamma_j^* = \int_{t_{n-1}}^{t_n} s_n s_{n-1} \cdots s_{n-j+1}\, dt \Big/ j!\, h^{j+1}$$

$$= (-1)^j \int_0^1 \frac{(-s+1)(-s)(-s-1)\cdots(-s-j+2)}{j!}\, ds$$

$$= (-1)^j \int_0^1 \binom{1-s}{j}\, ds \qquad (\gamma_0^* = 1).$$

The neglected term can be integrated, in the scalar case, to give

$$d_n = \gamma_k^* h^{k+1} y^{(k+1)}(\xi), \quad \xi \in (t_{n-k+1}, t_n).$$

We can estimate the local error, in the general case, by the quantity $\gamma_k^* h \nabla^k \dot{y}_n$.

The linear multistep formula obtained here is

$$y_n = y_{n-1} + h \sum_{j=0}^{k-1} \gamma_j^* \nabla^j \dot{y}_n .$$

This is the implicit Adams, or Adams-Moulton, formula of order k. It also can be written in the standard form

$$y_n = y_{n-1} + h \sum_{i=0}^{k-1} \beta_i^* \dot{y}_{n-i} \quad (\kappa_1 = 1, \ \kappa_2 = k-1),$$

but then the β_i^* depend on k, while the γ_i^* do not. We have

$$\beta_0^* = \sum_{j=0}^{k-1} \gamma_j^* = \gamma_{k-1} > 0$$

(the identity between the γ's being a consequence of their defining equations, proved in Section (c)). Hence these are all implicit methods. They are (k-1)-step methods, not k-step.

The values of coefficients γ_j and γ_j^* can be found in the text (tables on p. 108 and p. 113). One observation to be made from these is that for all $k > 1$, $|\gamma_k^*| < |\gamma_k|$. This means that at a given order, the local error is smaller for the implicit method that for the explicit one. There are also other reasons for preferring the implicit Adams methods, which will be clearer in Section (e), regarding stability.

The problem of starting up either of the Adams methods is rather easily solved when backward differences are used. One simply uses the order

l method on the first step, constructs $\nabla \dot{y}_1$ from the results, then uses the order 2 method for the second step, etc. The order can be built up to a given k in k steps in this way.

In the case of the implicit Adams methods, the predicting and correcting are generally done with the help of the explicit Adams method. Thus for the order k method, one could write

$$y_{n(0)} = y_{n-1} + h \sum_{0}^{k-1} \gamma_i \nabla^i \dot{y}_{n-1}$$

for the prediction. Then the corrections, as done by functional iteration, would be written

$$y_{n(m+1)} = y_{n-1} + h \sum_{0}^{k-1} \gamma_i^* \nabla^i \dot{y}_{n(m)} \quad (m = 0, 1, \cdots),$$

where in $\nabla^i \dot{y}_{n(m)}$ we use $f(y_{n(m)}, t_n)$ in place of $\dot{y}_n$ (which is unknown). The latter equation can be simplified to

$$y_{n(m+1)} = y_{n(m)} + h \gamma_{k-1} \left[\dot{y}_{n(m)} - \dot{y}_{n(m-1)} \right] \quad (m \geq 1),$$

$$y_{n(1)} = y_{n(0)} + h \gamma_{k-1} \nabla^k \dot{y}_{n(0)}.$$

These methods are referred as the Adams-Bashforth-Moulton methods.

9. (c) Symbolic Derivations

The use of symbolic methods, also referred to as the calculus of operators, is a powerful tool in the development of ODE methods, and also in methods in interpolation, extrapolation, quadrature, and numerical differentiation. Many fundamental formulas in these areas can be

derived with relatively little effort by symbolic methods, where conventional derivations would be lengthy and tedious. Here these methods will be introduced and used only to derive the Adams formulas (both explicit and implicit), and Gear's implicit methods.

(1) Basic Operators. We consider a general function f(t) of a real variable and a fixed real number h (to be interpreted later as a step-size). We can apply to f the operator Δ and get a new function Δf whose values, by definition, are

$$\Delta f(t) = f(t+h) - f(t) . \tag{1}$$

This defines the forward difference operator Δ. We define also the backward difference operator ∇ (read "del") by

$$\nabla f(t) = f(t) - f(t-h) \tag{2}$$

An operator that is simpler than either of these is the increment operator E which replaces t by t + h in the argument of f:

$$E f(t) = f(t+h). \tag{3}$$

Simpler yet is the operator k, where k is any constant, given simply by multiplication by k:

$$k f(t) = k \cdot f(t) \tag{4}$$

In particular, 1 denotes the identity operator. Finally, we consider the differentiation operator D, defined by

$$D f(t) = f'(t) \tag{5}$$

always assuming that all derivatives exist as needed.

These operators - Δ, ∇, E, k, and D -- are the basic ones. We can construct others by taking certain products and sums of these. For example, $\nabla^2 = \nabla \cdot \nabla$ is given by

$$\begin{aligned}\nabla^2 f(t) &= \nabla\,(\nabla f)(t) \\ &= \nabla\,[f(t) - f(t-h)] \\ &= [f(t) - f(t-h)] - [f(t-h) - f(t-2h)] \\ &= f(t) - 2f(t-h) + f(t-2h).\end{aligned}$$

Similarly, the operator products ΔD and $E^2\nabla$ are defined, by successive application of the basic operators, to be

$$\Delta D f(t) = f'(t+h) - f'(t) = D\Delta f(t),$$

$$E^2 \nabla f(t) = f(t+2h) - f(t+h) = \nabla E^2 f(t).$$

We notice that $E^m E^n = E^{m+n}$, and likewise for the other operators. Moreover, we can also consider negative powers of E:

$$E^{-1} f(t) = f(t-h), \tag{6}$$

i.e., the function which, when operated on by E, gives f again.

(2) Operator Equations. From (1) and (2), we can express Δ and ∇ in terms of E, for

$$\Delta f(t) = E f(t) - f(t) = (E-1) f(t)$$

$$\nabla f(t) = f(t) - E^{-1} f(t) = (1 - E^{-1}) f(t).$$

When two operators always give the same result, we can identify the operators as such. That is, we write

$$\Delta = E - 1 \tag{7}$$

$$\nabla = 1 - E^{-1} \tag{8}$$

We are now considering operators as independent quantities, separate from the functions to which they apply and which they produce. In doing so, we will perform algebraic manipulations on them, and we will use the fact that they commute and satisfy the associative law and distributive law. I.e., they satisfy the usual laws that hold for real numbers. This will be carried to the extent of writing operator expressions which cannot be defined explicitly in terms of their action on f(t), as the basic operators were. When this happens, some additional justification is needed to give the equations rigorous meanings. However, such rigor is beyond our scope here.

To illustrate the manipulations we will perform, we can rewrite (8) as

$$E = \frac{1}{1-\nabla} . \tag{9}$$

To obtain another basic operator equation we consider an infinite Taylor series:

$$f(t+h) = f(t) + h f'(t) + \frac{h^2}{2!} f''(t) + \cdots$$

$$= \left(1 + hD + \frac{h^2 D^2}{2!} + \cdots\right) f(t).$$

Thus from (3) we write

$$E = 1 + hD + \frac{h^2 D}{2!} + \cdots .$$

This series is, formally, just the infinite series for e^{hD}, and so we write

$$E = e^{hD}, \tag{10}$$

or, extending the formalism further yet,

$$D = h^{-1} \log E . \tag{11}$$

Here log E is an operator (namely hD) which we would be hard put to define from first principles (without reference to (10)). It is not the operator whose outcome on application to f(t) is log[Ef(t)].

(3) Implicit Adams Methods. Let us apply these symbolic operators to the problem of deriving the implicit Adams methods. Here we take f(t) = $\dot{y}$(t), and write

$$y(t_n) - y(t_{n-1}) = \int_{t_{n-1}}^{t_n} f(s)\,ds , \tag{12}$$

which we wish to approximate as a linear combination of $f_n = f(t_n)$, $f_{n-1} = f(t_{n-1})$, etc. If we let $t_n = t$ and $t_{n-1} = t_n - h$, then the above is $\nabla y(t)$, and $f(t) = Dy(t)$. Multiplying by D^{-1} (an operator which has not been explicitly defined, but would correspond to taking the indefinite integral) we get

$$\nabla y(t) = \nabla D^{-1} f(t) \tag{13}$$

If the operator ∇D^{-1} in (13) can be expressed in terms of ∇ alone, say in power series form, then (13), evaluated at t_n, would be the desired formula.

To carry this out, we use (11) and (9) to write

$$D^{-1} = h / \log E$$
$$= h / \log \left(\frac{1}{1-\nabla} \right),$$

and hence

$$\nabla D^{-1} = h \nabla / \log \left(\frac{1}{1-\nabla} \right) \qquad (14)$$

The operator in (14) has a formal power series in ∇, because the function of a complex variable

$$G^*(z) = z / \log \left(\frac{1}{1-z} \right)$$

has a power series in z, which is rigorously valid for $|z| < 1$. This series is

$$G^*(z) = z / \log \left(\frac{1}{1-z} \right) = \sum_0^\infty \gamma_i^* z^i , \qquad (15)$$

where $\gamma_0^* = 1$, $\gamma_1^* = -\frac{1}{2}$, ... G^* is called the generating function for the sequence γ_i^*. Formally, we then have the operator identity

$$\nabla D^{-1} = h \sum_0^\infty \gamma_i^* \nabla^i .$$

Substituting this into (13) and evaluating at $t = t_n$ as in (12), we obtain

$$\nabla y(t) = h \sum_0^\infty \gamma_i^* \nabla^i f(t),$$

$$y(t_n) - y(t_{n-1}) = h \sum_0^\infty \gamma_i^* \nabla^i f_n . \tag{16}$$

The implicit Adams methods arise by taking only the first k terms in (16), giving the kth order implicit formula

$$y_n - y_{n-1} = h \sum_0^{k-1} \gamma_i^* \nabla^i \dot{y}_n . \tag{17}$$

The error committed in doing this is a series which can be approximated by the first neglected term,

$$h \gamma_k^* \nabla^k \dot{y}_n .$$

(4) Explicit Adams Methods. To obtain the explicit analogue of (17), we want to approximate (12) by a combination of the quantities $\nabla^i f_{n-1}$ instead of $\nabla^i f_n$. But $f_{n-1} = f(t_{n-1}) = f(t_n - h) = E^{-1} f(t_n)$. Hence in (13) we should operate on $E^{-1} f$ instead of f:

$$\nabla y(t) = (\nabla D^{-1} E) E^{-1} f(t) . \tag{18}$$

Now the operator $\nabla D^{-1} E$ must be computed in terms of ∇. But this requires only an additional use of (9):

$$\nabla D^{-1} E = \nabla \cdot \left[h / \log E \right] \frac{1}{1-\nabla}$$

$$= \frac{h \nabla}{1-\nabla} \Big/ \log\left(\frac{1}{1-\nabla}\right)$$

Again we express this in series form by use of a generating function:

$$G(z) = \frac{z}{1-z} \Big/ \log\left(\frac{1}{1-z}\right) = \sum_0^\infty \gamma_i z^i , \tag{19}$$

where $\gamma_0 = 1$, $\gamma_1 = \frac{1}{2}$, ... Thus we write

$$\nabla D^{-1} E = h \sum_0^\infty \gamma_i \nabla^i ,$$

substitute into (18), and evaluate at t_n to get

$$y(t_n) - y(t_{n-1}) = h \sum_0^\infty \gamma_i \nabla^i f_{n-1} . \tag{20}$$

The explicit Adams methods arise by retaining only the first k terms above, giving the kth order formula

$$y_n - y_{n-1} = h \sum_0^{k-1} \gamma_i \nabla^i \dot{y}_{n-1} , \tag{21}$$

with an error that is approximately

$$h \gamma_k \nabla^k \dot{y}_{n-1} .$$

(5) Coefficient Relations. The generating functions (15) and (19) provide a convenient way to generate the needed Adams coefficients. For example, (19) may be rewritten

$$\left(z + \frac{z^2}{2} + \frac{z^3}{3} + \cdots\right)\left(\gamma_0 + \gamma_1 z + \cdots\right) = z + z^2 + z^3 + \cdots ,$$

using the familiar series for $\log \frac{1}{1-z}$ and $\frac{1}{1-z}$. Taking coefficients of z^{m+1}, this yields

$$\gamma_m + \frac{\gamma_{m-1}}{2} + \frac{\gamma_{m-2}}{3} + \cdots \frac{\gamma_0}{m+1} = 1 . \tag{22}$$

This relation can be used recursively to get the γ_i from $\gamma_0 = 1$.

If the γ_i are generated as above, the γ_i^* can be easily obtained by noting that

$$G^*(z) = (1-z)\,G(z)$$

$$\sum_0^\infty \gamma_i^* z^i = (1-z) \sum_0^\infty \gamma_i z^i .$$

Equating the coefficients of z^m gives

$$\gamma_m^* = \gamma_m - \gamma_{m-1} . \tag{23}$$

An alternative approach is to rewrite (15) as

$$\left(z + \frac{z^2}{2} + \cdots\right)\left(\gamma_0^* + \gamma_1^* z + \cdots\right) = z ,$$

and obtain, for $m > 0$,

$$\gamma_m^* + \frac{\gamma_{m-1}^*}{2} + \cdots + \frac{\gamma_0^*}{m+1} = 0 .$$

This can be used to generate the γ_i^* from $\gamma_0^* = 1$. Then the γ_i can be generated by rewriting (23) as

$$\gamma_m = \sum_{i=0}^{m} \gamma_i^* . \tag{24}$$

(6) Gear Methods. The formulas of Gear are implicit ones in which $\dot{y}_n$ is set equal to a linear combination of $y_n, y_{n-1}, \ldots, y_{n-k}$. Hence they can be derived by writing the D operator as a series in ∇. This is easily obtained from (9) and (11):

$$hD = \log \frac{1}{1-\nabla} = \sum_{i=1}^{\infty} \frac{1}{i} \nabla^i . \tag{25}$$

By retaining only k terms in this series and applying to $y(t)$ at $t = t_n$, we obtain the k-step formula

$$h\dot{y}_n = \sum_{i=1}^{k} \frac{1}{i} \nabla^i y_n \quad . \tag{26}$$

The equation has an error that is approximated by the first neglected term,

$$\frac{1}{k+1} \nabla^{k+1} y_n \simeq \frac{h^{k+1} y_n^{(k+1)}}{k+1} \quad .$$

Thus the method given by (26) is of order k.

If (26) is written in the standard form

$$y_n = \sum_{i=1}^{k} \alpha_i y_{n-i} + h\beta_0 \dot{y}_n \, , \tag{27}$$

one must divide (26) by the coefficient of y_n, which is

$$\sum_{i=1}^{k} \frac{1}{i} \quad .$$

Hence

$$\beta_0 = 1 \Big/ \sum_{i=1}^{k} \frac{1}{i} \quad . \tag{28}$$

The α_i can be obtained by expressing the remaining terms in (26) in terms of the y_{n-i}. To do this, we use (8) and the binomial theorem to get

$$\nabla^i = (1 - E^{-1})^i = \sum_{j=0}^{i} (-1)^j \binom{i}{j} E^{-j} \tag{29}$$

Substituting into (26) and rearranging,

$$h\dot{y}_n = \sum_{i=1}^{k} \sum_{j=0}^{i} \frac{1}{i} (-1)^j \binom{i}{j} E^{-j} y_n$$

$$= \sum_{j=0}^{k} \left[\sum_{\substack{i=j \\ i \geq 1}}^{k} \frac{1}{i} (-1)^j \binom{i}{j} \right] y_{n-j} .$$

Identifying the coefficient in brackets as $-\alpha_j/\beta_0$, we obtain

$$\alpha_j = \beta_0 (-1)^{j+1} \sum_{\substack{i=j \\ i \geq 1}}^{k} \frac{1}{i} \binom{i}{j} . \qquad (30)$$

The relations (30), together with (28), provide the coefficients for Gear's methods in the form (27). (In particular they give $\alpha_0 = -1$.)

9. (d) Order, associated polynomials, asymptotic error

The subjects of order and local error relate to the question of how closely a true solution function y(t) satisfies the linear multistep formula under consideration, which we write, as before,

$$y_n = \sum_{1}^{K_1} \alpha_i y_{n-i} + h \sum_{0}^{K_2} \beta_i \dot{y}_{n-i} .$$

Thus for a given function y(t) we define an operator L_h on y(t) by

$$L_h(y(t)) = -y(t) + \sum_{1}^{K_1} \alpha_i y(t-ih) + h \sum_{0}^{K_2} \beta_i \dot{y}(t-ih).$$

This can be computed for any y(t) as long as $\dot{y}(t)$ exists. The local accuracy question is now that of how small $L_h(y(t))$ is. In specific examples (Section (a)) we saw that by choosing the α_1 and β_1 properly, it is possible to make

$L_h(y(t)) = 0$ identially when $y(t)$ is a polynomial of a certain degree. That same kind of analysis will, in the general case, show that for some integer r,

$$L_h(y(t)) = O(h^{r+1})$$

as $h \to 0$, for $y(t)$ for which $y^{(r+1)}(t)$ is continuous. This is done by inserting Taylor series for the various terms and collecting like powers of h. For the case $K_1=1$, $K_2=2$, we got $r=3$ for the unique choice of coefficients which makes r maximal. The terms in h^0, h^1, h^2, and h^3 in $L_h(y(t))$ in that example have coefficients that vanish by virtue of the coefficient conditions.

Returning to the general situation, for any given value of r and of the coefficients, we can write, by direct use of Taylor series,

$$L_h(y(t)) = \sum_{q=0}^{r+1} C_q h^q y^{(q)}(t) + O(h^{r+2}),$$

if $y(t)$ has continuous derivatives up to order $r+2$. The coefficients C_q depend on the α_i and β_i, but not on h or $y(t)$. If we include the term $-y(t)$ into the first sum, with a coefficient $\alpha_0 = -1$, then we find that

$$C_q = \sum_{i=0}^{K_1} (-i)^q \alpha_i / q! + \sum_{i=0}^{K_2} (-i)^{q-1} \beta_i / (q-1)!$$

$$C_0 = \sum_{0}^{K_1} \alpha_i .$$

We can now define order formally, and restate the definition in terms of the C_q:

Definition: The order of the linear multistep method with coefficients α_i and β_i is the largest integer r for which

$$L_h(y(t)) = C_{r+1} h^{r+1} y^{(r+1)}(t) + O(h^{r+2})$$

as $h \to 0$ for fixed t, for any y(t) with $y^{(r+1)}$ continuous. If a method has order r, and $y^{(r+2)}$ is also continuous, then we can carry the Taylor series to the h^{r+1} terms, and it must reduce to

$$L_h(y(t)) = C_{r+1} h^{r+1} y^{(r+1)}(t) + O(h^{r+2}).$$

with $C_{r+1} \neq 0$. In other words, the order is the largest integer r for which

$$C_0 = C_1 = \cdots = C_r = 0, \quad C_{r+1} \neq 0.$$

The r+1 equations $C_q = 0$ $(q \leq r)$ necessary for order r involve the K_1+K_2+1 coefficients α_i $(1 \leq i \leq K_1)$ and β_i $(0 \leq i \leq K_2)$. So one would expect to be able to solve for those coefficients with $r=K_1+K_2$, or less. In fact we can do that, but the resulting method is not automatically a good one. The reason for this has to do with stability, and will be discussed in Section (e).

There is an even simpler way to express the order conditions. For this we need only consider the simple case $y(t)= e^{\lambda t}$, $\dot{y}= \lambda y= \lambda e^{\lambda t}$. We can compute L_h for this function:

$$L_h(e^{\lambda t}) = \sum_0^{K_1} \alpha_i e^{\lambda(t-ih)} + h\lambda \sum_0^{K_2} \beta_i e^{\lambda(t-ih)}$$

$$= e^{\lambda(t-hK)} \left[\sum_0^{K_1} \alpha_i e^{\lambda h(K-i)} + h\lambda \sum_0^{K_2} \beta_i e^{\lambda h(K-i)} \right].$$

We define two polynomials of degree K:

$$\rho(\xi) = \sum_{i=0}^{K_1} \alpha_i \xi^{K-i},$$

$$\sigma(\xi) = \sum_{i=0}^{K_2} \beta_i \xi^{K-i} \qquad [K \equiv \max(K_1, K_2)],$$

called the associated polynomials for the method. Then we have simply

$$L_h(e^{\lambda t}) = e^{\lambda t} e^{-\lambda h \kappa} [\rho(e^{\lambda h}) + h\lambda \sigma(e^{\lambda h})].$$

Now if r is the order of the method, and we let $h \to 0$, then, by definition,

$$L_h(e^{\lambda t}) = C_{r+1} h^{r+1} y^{(r+1)}(t) + O(h^{r+2})$$
$$= C_{r+1} h^{r+1} \lambda^{r+1} e^{\lambda t} + O(h^{r+2}).$$

Combining this with the previously computed expression, we must have

$$\rho(e^{h\lambda}) + h\lambda \sigma(e^{h\lambda}) = e^{\lambda h \kappa} [C_{r+1} (h\lambda)^{r+1} + O(h^{r+2})]$$
$$= C_{r+1} (h\lambda)^{r+1} + O(h^{r+2})$$

Now make a change of variables to $z = e^{h\lambda} - 1$. Then $z = h\lambda + O(h^2)$ as $h \to 0$, and we have

$$\rho(1+z) + \log(1+z)\sigma(1+z) = C_{r+1} z^{r+1} + O(z^{r+2})$$

as $z \to 0$, with $C_{r+1} \neq 0$.

This asymptotic relation on the associated polynomials is an alternate characterization of the order r. It is more convenient in that only one function is involved, not a whole sequence of coefficient formulas. We simply write down the polynomials ρ and σ, expand $\rho(1+z) + \log(1+z)\sigma(1+z)$, which is an analytic function of z near z=0, as $\sum_0^\infty D_q z^q$, and evaluate the first non-vanishing term, $C_{r+1} z^{r+1}$. This calculation uses the series

$\log(1+z) = z - z^2/2 + z^3/3 - z^4/4 + \ldots$.(The D_q will not all be the same as the C_q in the expansion of L_h, however.)

It is clear that the original order conditions imply the above condition on ρ and σ. To prove the converse implication, we suppose that the latter condition holds, and by substituting $e^{h\lambda} = 1 + z$, get

$$L_h(e^{\lambda t}) = e^{\lambda t} C (h\lambda)^{r+1} + O(h^{r+2})$$

for a constant $C \neq 0$. But the general expansion of $L_h(y(t))$ gives, using $y^{(q)} = \lambda^q e^{\lambda t}$,

$$L_h(e^{\lambda t}) = \sum_{q=0}^{\infty} C_q h^q \lambda^q e^{\lambda t}.$$

In order for these to agree, we must have $C_0 = C_1 = \ldots = C_r = 0$, and $C_{r+1} = C$. This proves the equivalence of the two characterizations of order.

The condition of consistency is defined as the condition $C_0 = C_1 = 0$, or

$$\sum_{1}^{K_1} \alpha_i = 1, \qquad \sum_{1}^{K_1} i\alpha_i = \sum_{0}^{K_2} \beta_i .$$

The equivalent conditions in terms of ρ and σ are

$$\rho(1) = 0, \qquad \rho'(1) + \sigma(1) = 0.$$

Clearly consistency means that the order is at least 1. This is a requirement that is invariably imposed on any method to be used.

The associated polynomials ρ, σ are an aid to analyzing a given linear multistep method, in determining the order r and the constant C_{r+1}. Together, these numbers describe the local error of the method and thus its accuracy. But ρ, σ are also an aid in deriving methods. For example, if one is given only the β_i of a method, so that one has $\sigma(\xi)$, then one can get the α_i

for which the order is maximal (or is any smaller given value) by writing down $\rho(\xi)$ from

$$\rho(\xi) + \log \xi \, \sigma(\xi) = O\left[(\xi-1)^{r+1}\right].$$

Similarly, if $\rho(\xi)$ is given, $\sigma(\xi)$ can be computed. Two examples of this process are given in the text (pp. 119-120): If $\sigma(\xi) = \frac{3}{2}\xi - \frac{1}{2}$, the 2-step method of maximal order must have, by an easy calculation, $\rho(\xi) = \xi - \xi^2$, leading to the explicit Adams method of order 2. If $\rho(\xi) = \xi - \xi^2$ is given, then the 2-step method of maximal order can be computed to have $\sigma(\xi) = \frac{5}{12}\xi^2 + \frac{2}{3}\xi - \frac{1}{12}$, corresponding to the implicit Adams method of order 3.

We now consider in greater detail the local error of a linear multistep method.

Definition: For the method

$$y_n = \sum_{1}^{K_1} \alpha_i \, y_{n-i} + h \sum_{0}^{K_2} \beta_i \, \dot{y}_{n-i}, \qquad \dot{y}_j = f(y_j, t_j),$$

the local error is the difference

$$d_n = y_n - y(t_n),$$

where $y(t)$ is a true solution of the ODE, and y_n is calculated by the method with $y_{n-i} = y(t_{n-i})$ for $1 \le i \le K$.

In other words, when all past values needed are exact, the error in the computed y_n is the local error. (The sign of d_n differs from that used in Section (b).)

For an explicit method ($\beta_0 = 0$), it is clear that

$$d_n = L_h(y(t))\big|_{t=t_n},$$

by the definition of L_h. However, when $\beta_0 \neq 0$, these two quantities are different, because the L_h expression involves the term $h\beta_0\dot{y}(t_n)$ while

the formula for y_n involves $h\beta_0 f(y_n,t_n)$. These two differ, but only by $O(h^{r+2})$, while the quantities themselves are $O(h^{r+1})$. More specifically, by writing out the formula for y_n, we have

$$d_n = \sum_{1}^{K_1} \alpha_i y_{n-i} + h \sum_{1}^{K_2} \beta_i \dot{y}_{n-i} + h\beta_0 f(y_n,t_n) - y(t_n)$$
$$= L_h(y)\big|_{t_n} + h\beta_0 \left[f(y(t_n)+d_n, t_n) - f(y(t_n), t_n) \right].$$

If we solve this implicit equation for d_n, we will get

$$d_n = L_h(y(t))\big|_{t=t_n} + O(h^{r+2}).$$

Thus for purposes of analyzing the asymptotic nature of local error, we may as well use L_h instead of the true local error d_n. (This situtation was discussed for the case of the implicit Euler method in Chapter 4, but with a different notation: ϵ_n for the local error in y_{n+1}, and d_n for $L_h(y)\big|_{t_{n+1}}$). We prefer to use L_h because it is given explicitly, while d_n, in the implicit case, is not. On the other hand, d_n is the more meaningful quantity, being the actual error in y_n, when past values are exact, and when we ignore roundoff and iteration errors.

In addition to looking at the individual errors, we can look at their accumulation into global errors. This was done for the Euler methods earlier. If $y(t)$ is the fixed solution of the given initial value problem, the global errors $e_n=y_n-y(t_n)$ can be shown to satisfy a recursion formula. This formula approximates the one corresponding to the solution of another ODE. The result is

$$e_n = h^r \delta(t_n) + O(h^{r+1}),$$

where $\delta(t)$ is a function given by the ODE

$$\dot{\delta}(t) = f_y(t)\delta(t) + \left(C_{r+1} / \sum_0^{K_2} \beta_i\right) y^{(r+1)}(t)$$

with the initial value $\delta(0) = 0$. The details are given in the text (pp. 204-205).

9. (e) Stability and convergence; root conditions

In this section, we look even more closely at the truncation errors, and how they can build up during the numerical solution of the problem.

To illustrate what can go wrong with a method, we consider the example of Milne's method. This method has $K_1 = K_2 = 2$, and the unique choice of the 5 coefficients which make the method have order $r = K_1 + K_2 = 4$. The formula is

$$y_n = y_{n-2} + \frac{h}{3}(\dot{y}_n + 4\dot{y}_{n-1} + \dot{y}_{n-2}),$$

and is obtained by applying Simpson's rule for $\int_{t_{n-1}}^{t_n} \dot{y}(t)dt$. The associated polynomials are

$$\rho(\xi) = -\xi^2 + 1, \qquad \sigma(\xi) = \frac{1}{3}\xi^2 + \frac{4}{3}\xi + \frac{1}{3},$$

and

$$\rho(1+z) + \log(1+z)\sigma(1+z) = (-2z - z^2)$$
$$+ \left(z - \frac{z^2}{2} + \frac{z^3}{3} - \frac{z^4}{4} + \frac{z^5}{5} - \cdots\right)\left(2 + 2z + \frac{1}{3}z^2\right) = \frac{1}{90}z^5 + \cdots.$$

So we expect the local error to be approximately $\frac{1}{90}h^5 y^{(5)}$.

Now consider applying Milne's method to $\dot{y} = \lambda y$, with some fixed h. The formula for step n is

$$y_n = y_{n-2} + \frac{h\lambda}{3}(y_n + 4y_{n-1} + y_{n-2}),$$

$$\left(1 - \frac{h\lambda}{3}\right)y_n = \frac{4h\lambda}{3}y_{n-1} + \left(1 + \frac{h\lambda}{3}\right)y_{n-2}.$$

This is a constant-coefficient linear recursion equation, and it is known that such an equation has solutions of the form

$$y_n = A\xi_1^n + B\xi_2^n$$

for constants A, B, ξ_1, ξ_2. If we try the sequence $y_n = \xi^n$ as a solution, for some $\xi \neq 0$, we find that ξ must satisfy

$$-(1 - h\lambda/3)\xi^n + \frac{4h\lambda}{3}\xi^{n-1} + (1+h\lambda/3)\xi^{n-2} = 0,$$

$$(-1+h\lambda/3)\xi^2 + \frac{4h\lambda}{3}\xi + (1+h\lambda/3) = 0.$$

To be specific, suppose that $\lambda = -1$ and $h = .1$, and the initial value is $y_0 = 1$ at $t_0 = 0$, so that the solution is $y = e^{-t}$. Then the above equation for ξ is (roughly)

$$-1.03\xi^2 - .133\xi + .97 = 0,$$

and its two roots are

$$\xi_1 \approx .90483737, \qquad \xi_2 \approx -1.034.$$

Now both ξ_1^n and ξ_2^n are solutions of the recursion equation, and so is any linear combination $A\xi_1^n + B\xi_2^n$. We can choose A and B to match the initial conditions. Since the step number is 2, we would need to have a second starting value to get the method going. Suppose we choose the exact value

$$y_1 = y(t_1) = e^{\lambda h} = e^{-.1} \approx .90483742.$$

To make the solution correct at n = 0, we must have

$$A + B = 1,$$

and to make it correct at n = 1, we must have

$$A\xi_1 + B\xi_2 = e^{-.1}.$$

We can solve these equations for A and B and get

$$B = (e^{-.1} - \xi_1)/(\xi_2 - \xi_1) \approx -2.5 \cdot 10^{-8}$$
$$A = 1 - B \approx 1 + 2.5 \cdot 10^{-8}.$$

Thus the numerical solution obtained for this problem with the given method and starting values is

$$y_n = (1 + 2.5 \cdot 10^{-8})(.9048)^n - 2.5 \cdot 10^{-8}(-1.034)^n,$$

provided we ignore machine roundoff (numerical constants given only approximately).

Examining this solution carefully, we see first that the first term is approximately

$$\xi_1^n \approx (e^{-.1})^n = e^{-hn} = e^{-t_n} = y(t_n).$$

This we expect, since the first term dominates, at least for small n. But eventually the second term will dominate. For n = 100, the two terms are about $5 \cdot 10^{-5}$ and $-8 \cdot 10^{-7}$, or the second term is more than 1% of the total. For $n > 130$ or so, the second term is larger than the first in magnitude, and it oscillates in sign. This problem can be made even worse by the introduction of roundoff errors, and of a less accurate value of y_1 to start up. (See tables on p. 123 of the text.)

The reason for this bad behavior is the fact that the root ξ_2 has a magnitude exceeding 1, so that its powers grow unboundedly. If we had gotten $|\xi_2| < 1$, this behavior would not occur.

These ideas extend easily for the general linear multistep method. Such a method applied to the ODE $\dot{y}=\lambda y$ gives

$$\sum_0^{K_1} \alpha_i\, y_{n-i} + h\lambda \sum_0^{K_2} \beta_i\, y_{n-i} = 0.$$

This difference equation will have a solution $y_n=\xi^n$ if ξ satisfies the equation

$$\sum_0^{K} (\alpha_i + h\lambda\beta_i)\, \xi^{n-i} = 0 \qquad (K = \max(K_1, K_2)),$$

or, factoring out ξ^{n-K},

$$\rho(\xi) + h\lambda\, \sigma(\xi) = 0.$$

For a given $h\lambda$, this is a polynomial equation of degree K, and has K roots $\xi_1, \xi_2, \ldots, \xi_K$. If these ξ_j are distinct, then the general solution of the difference equation is an arbitrary linear combination of the ξ_j^n:

$$y_n = \sum_{j=1}^{K} c_j\, \xi_j^{\,n}.$$

If there are multiple roots, the general solution is slightly different. For example, if ξ_1 is a double root, or $\xi_2=\xi_1$, then the general solution has $c_1\,\xi_1^{\,n} + c_2 n\,\xi_1^{\,n}$ in place of $c_1\,\xi_1^{\,n} + c_2\,\xi_2^{\,n}$. Higher multiplicities lead to higher powers of n.

In any case, the general solution has K coefficients, and if we specify the values $y_0, y_1, \ldots, y_{K-1}$ needed to start up, then the coefficients are determined uniquely. So, ignoring roundoff, we again know the calculated solution values y_n completely.

The roots ξ_j of $\rho + h\lambda\sigma$ are clearly functions of $h\lambda$. If h is small, we can get an idea of their value by looking at ρ alone. We already know one of the roots of ρ, because the 0^{th} order condition on the method is

$$\rho(1) = \sum_0^{K_1} \alpha_i = 0.$$

Let us call this root $\xi_1 = 1$, at $h\lambda = 0$. For $h\lambda \neq 0$, ξ_1 will vary from 1, and in fact it will approximate $e^{h\lambda}$ according to

$$\xi_1 = e^{h\lambda} + \frac{C_{r+1}}{\sigma(1)} (h\lambda)^{r+1} + O(h^{r+2}).$$

Such a relation must hold if the quantity ξ_1^n is to approximate $e^{\lambda t_n = e^{h\lambda n}}$ with a local error of $O(h^{r+1})$. The root $\xi_1 = \xi_1(h\lambda)$ is called the principal root associated with the method. If the coefficient of ξ_1^n in y_n is $c_1 \approx y(0)$, then the first term in y_n is

$$c_1 \xi_1^n \approx y(0) e^{\lambda t_n} = y(t_n).$$

If the problem is started with sufficiently accurate starting values, then c_1 will in fact be close to $y(0)$.

All the other roots ξ_j are called extraneous or parasitic roots. They repesent error terms that are unavoidably present because of the multistep nature of the method. We can only try to minimize this source of error by making the extraneous ξ_j smaller than 1 in magnitude. Then the terms $c_j \xi_j^n$ will all decay to 0 as n grows, so that the first term will dominate.

Definition: A linear multistep method is called strongly stable if all the roots of $\rho(\xi) = 0$ are in the unit disk $|\xi| < 1$, except for a simple root at $\xi = 1$.

While this condition of strong stability is enough to obtain stable behavior for the method, one can prove stability with slightly weaker conditions.

Definition: A linear multistep method is called weakly stable if all the roots of $\rho(\xi) = 0$ are in the disk $|\xi| \leq 1$, and on the circle $|\xi| = 1$ there are two or more roots, all of them simple.

In other words, additional roots are allowed on the boundary of the unit disk for weak stability, as long as they are not multiple roots, while for strong stability only the principal root $\xi = 1$ is allowed there.

Definition: A linear multistep method satisfies the root condition if it is either strongly or weakly stable, i.e. if all roots of $\rho(\xi) = 0$ lie in $|\xi| \leq 1$, and there are no multiple roots on $|\xi| = 1$.

The following theorem states the relationship between these properties of the roots associated with a method and the numerical behavior of the method. Recall the concept of stability: A method is stable if perturbations in the starting values result in bounded perturbations in the calculated y_n at the end of a fixed t interval. Recall also the concept of convergence: A method is convergent if the solution y_n calculated at the end of a fixed interval converges to the true solution $y(t)$ as $h \to 0$ and as the errors in the starting values all tend to 0.

Theorem: If a consistent linear multistep method satisfies the root condition (i.e. is strongly or weakly stable), then it is stable and convergent. Conversely, if it is stable and convergent, then the root condition must hold.

We will not prove this theorem here, as that would take us too far afield. (An easily accessible proof (first part only) appears in Henrici, Ref. 15, Section 5.3.)

The following are some examples of methods and their properties relative to the root condition.

(1) Milne's method has $\rho(\xi) = 1 - \xi^2$. The roots are $\xi_1 = 1$ and $\xi_2 = -1$. Thus the method is only weakly stable.

(2) The 3rd order explicit method

$$y_n = -4y_{n-1} + 5y_{n-2} + h(4\dot{y}_{n-1} + 2\dot{y}_{n-2})$$

is the explicit 2-step method of maximal order. It has $\rho(\xi) = -\xi^2 - 4\xi + 5 = (1-\xi)(5+\xi)$. So $\xi_2 = -5$, and the method is unstable.

(3) Hamming's method has $\rho(\xi) = -\xi^3 + \frac{9}{8}\xi^2 - \frac{1}{8} = (1-\xi)(\xi^2 - \frac{1}{8}\xi - \frac{1}{8})$. The roots are $\xi_1 = 1, \xi_2 \simeq .30, \xi_2 \simeq -.42$, and so the method is strongly stable.

(4) The Adams methods, either explicit or implicit, have the form

$$y_n = y_{n-1} + h\sum_0^K \beta_i \dot{y}_{n-i}.$$

Thus $\rho(\xi) = -\xi^K + \xi^{K-1}$, and the roots are $\xi_1 = 1, \xi_2 = \ldots = \xi_K = 0$. So these methods are strongly stable. In fact, no other method has better stability properties than this, as far as root conditions on ρ are concerned.

These examples show why a linear multistep method cannot be judged only from its order and local error properties. All of the above are consistent methods of various orders, and all may appear satisfactory at first sight. But their stability and convergence properties differ widely, from very good to very bad.

The question that now arises is: how well can one do in constructing accurate but stable linear multistep methods. Making the order have its maximal value $r = K_1 + K_2$ does <u>not</u> always result in a worthwhile method,

because of stability. On the other hand, the implicit Adams methods have order $r = K_1 + K_2 = 1 + K$, and they are very stable. The following theorem settles this question in general. It will not be proved here either.

Theorem (Dahlquist): For an order r, K-step, linear multistep method, the method can be stable (weakly or strongly) only if $r \leq K + 1$ for odd K, or $r \leq K + 2$ for even K. The method can be strongly stable only if $r \leq K + 1$.

That is, if we reject the weakly stable methods as unacceptable, then the best we can do for a K-step method is order K + 1. (This is achieved by the implicit Adams methods, among others.) This is to be compared with the value $r = K_1 + K_2$, which is 2K if $K_1 = K_2$, achievable without regard for stability.

The properties of the roots of $\rho(\xi)$ really give only the properties of the method in the limit as $h \to 0$. This is not the entire story, because h is never really 0, and often is not close enough to 0 to infer the needed information from that limiting case. For the test equation $\dot{y} = \lambda y$, the roots ξ_j of $\rho(\xi) + h\lambda\sigma(\xi)$ can be computed for any given $h > 0$, and these are the roots that determine the numerical behavior of the method. As stated earlier, it can be argued that the situation for this test equation, while unrealistically simple, will indicate the situation for the general problem, where we take λ to be any of the eigenvalues of the Jacobian $\partial f/\partial y$.

It is impractical to describe the full set of roots $\xi_j(h\lambda)$ as functions of $h\lambda$, as $h\lambda$ ranges over the whole complex plane. We make the stability question much simpler by asking only for the values of $h\lambda$ that make all the $|\xi_j| \leq 1$.

Definition: The absolute stability region of a linear multistep method is the set

$$S_a = \{ h\lambda : |\xi_j(h\lambda)| \leq 1 \quad \text{for all } j \},$$

where ξ_j are the roots of $\rho + h\lambda\sigma$.

The notion of absolute stability given here is qualitatively consistent with the notion of bounded perturbations in the definition of stability. For the equation $\dot{y} = \lambda y$, subjected to a method for which $h\lambda \in S_a$, the computed values of y_n are linear combinations of the ξ_j^n, and so are the perturbations e_n in y_n resulting from a perturbation of the starting values:

$$e_n = \sum_{j=0}^{K} d_j \xi_j^n .$$

These perturbations all decay, or at least remain bounded (provided there are no multiple roots with $|\xi_j| = 1$), by virtue of $h\lambda \in S_a$. If $h\lambda$ lies in the interior of S_a, so that all the $|\xi_j| < 1$ strictly, then $e_n \to 0$ as $n \to \infty$. The context here, however, is that of a fixed h, and $n \to \infty$, rather than a fixed $t_n = t_0 + nh$, with $h \to 0$ and $n \to \infty$, as in the definition of stability.

If λ is in the left half-plane, or $\mathrm{Re}(h\lambda) < 0$, then we expect that $|\xi_1| < 1$, because $\xi_1 \approx e^{h\lambda}$, and $|e^{h\lambda}| = e^{\mathrm{Re}(h\lambda)} < 1$. So for sufficiently small values of $h\lambda$ at least, the term ξ_1^n is not a source of instability. But in general, especially when $|h\lambda|$ is sizeable, all of the ξ_j are potential sources of numerical instability. Thus the requirement $|\xi_j| \leq 1$ in S_a is applied to all the roots, ξ_1 included.

If $Re(h\lambda) > 0$, then $|e^{h\lambda}| > 1$ and the function $e^{\lambda t}$ is growing instead of decaying. In that case, we expect $|\xi_1| > 1$, and so we would not expect to have $h\lambda \in S_a$, although in many cases S_a does include part of the right half-plane. Rather than require that the $|\xi_j|$ be smaller than one, the concern here is that the principal term $c_1 \xi_1^n$ remains dominant over the others. To achieve this, we could ask that the extraneous roots satisfy $|\xi_j| \leq |\xi_1|$. This leads to the concept of relative stability:

Definition: The relative stability region of a linear multistep method is the set

$$S_r = \{h\lambda : |\xi_j| \leq |\xi_1| \text{ for all } j > 1\},$$

where $\xi_j = \xi_j(h\lambda)$ are the roots of $\rho + h\lambda\sigma$.

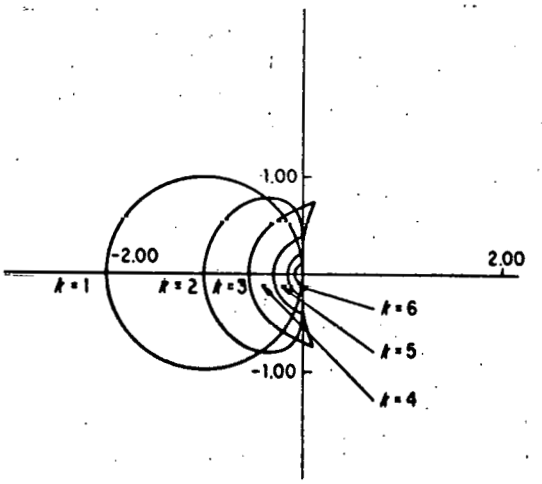
Thus for $h\lambda \in S_r$, the parasitic roots will not cause significant errors relative to the principal root, although the absolute errors associated with them may be considerable.

When $Re(h\lambda) > 0$, there is another consideration that enters in, besides that of relative stability. This is that h be chosen so that the method computes $y_n \approx y_0 e^{\lambda t_n}$ accurately. This issue is determined by the local truncation errors, and is not directly related to stability. It is instead the issue of how well ξ_1 approximates $e^{h\lambda}$, and not the size of the other ξ_j. Usually, the constraint on h imposed by this accuracy consideration is more restrictive than the constraint imposed by relative stability (i.e. by S_r).

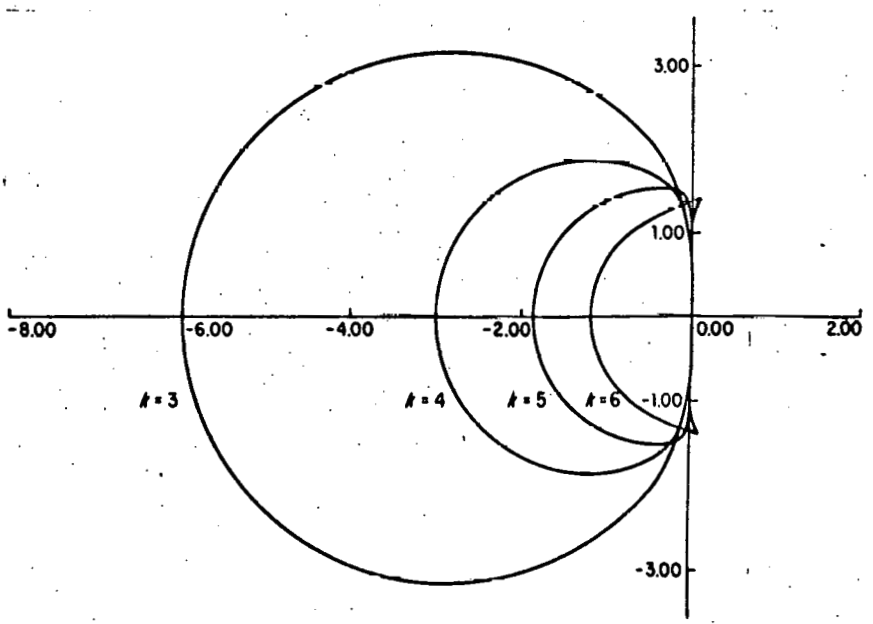
We conclude this section with some illustrations of absolute stability regions. The region S_a is relatively easy to compute for a given method, because its boundary is given by

$$\partial S_a = \{ h\lambda : \text{some } |\xi_j| = 1 \} = \left\{ h\lambda = \frac{-\rho(\xi)}{\sigma(\xi)} : \xi = e^{i\theta},\ 0 \le \theta < 2\pi \right\}.$$

Sketches of S_a for the explicit Adams methods are shown here (taken from Fig. 8.1, p. 131 in the text). Note that these regions lie largely in the left half-plane and that they tend to be smaller for the higher orders (k). The case k=1 is the explicit Euler method, for which S_a is the disk $|1 + h\lambda| \le 1$.

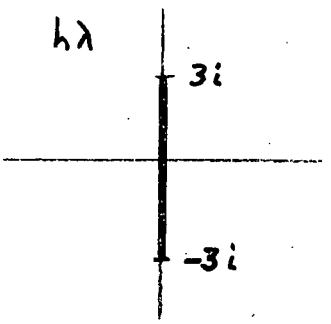


Sketches of S_a for the implicit Adams method are shown here for orders 3 to 6 (Fig 8.2 in text). The case k = 1 is the implicit Euler method, with $S_a = \{h\lambda : |1-h\lambda| \ge 1\}$. The case k = 2 is the trapezoid rule, whose absolute stability region is the left half plane. An important observation here is that for a



given order k, the region S_a in the explicit case is much smaller than that in the implicit case. This is an additional factor in favor of the implicit Adams, or Adams-Moulton, methods (the first being the smaller local error coefficients).

The absolute stability region of Milne's method turns out to be the vertical line segment from -3i to 3i on the imaginary axis of the comples $h\lambda$ plane. This fact is in conformity with unstable behavior of the method observed earlier. For virtually any value of $h\lambda$, one of the ξ_j here is larger than 1 in magnitude.



It should be noted that in computer codes which implement linear multistep methods, the stability regions are generally not computed or supplied in any direct way. That is, ODE codes do not compute eigenvalues λ and test the $h\lambda$ to see if they belong to S_a, for example. Such a computation would be too costly to be worthwhile. However, the instability that occurs if the $\mathrm{Re}(h\lambda) < 0$ and some $h\lambda \notin S_a$ is generally detected in an indirect way in ODE codes. What happens is that the resulting unstable behavior causes the local error estimates to become large. Thus if the order is r, these estimates would indicate the presence of a large $y^{(r+1)}$ (even though the true $y^{(r+1)}$ may not be as large as indicated), and the step size h is reduced as a result. Thus the error control tends to keep the $h\lambda$ in the appropriate

stability region, but only indirectly. If the code performs iterations for solution of an implicit equation, and makes a test for convergence of these iterations, that test may provide another indirect control on the stability of the method.

9. (f) Matrix formulations and multivalue methods

In this section we present some ways in which general linear multistep methods can be restated in a more compact notation. As a result of such reformulations, it is somewhat easier to understand the workings of the methods and to implement them. This material applies primarily to implicit methods. Details can be found in Ref. 18.

Write the general implicit linear multistep method as

$$y_n = \sum_{1}^{K_1} \alpha_i^* y_{n-i} + h \sum_{0}^{K_2} \beta_i^* \dot{y}_{n-i} \ .$$

We use asterisks on the coefficients to designate the given implicit method; coefficients without asterisks will be used for explicit methods. Recall from Section 7 (a) that the natural past history array that we would store is

$$\underline{y_n} = \begin{pmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n+1-K_1} \\ \hline h\dot{y}_n \\ \vdots \\ h\dot{y}_{n+1-K_2} \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ b \\ c \\ d \end{matrix}}\right\} K_1 \text{ rows} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} K_2 \text{ rows} \end{matrix}$$

This array has $L = K_1 + K_2$ rows and N columns, N being the length of each vector above. (We usually store the product $h\dot{y}_j$ instead of $\dot{y}_j$ because these products appear in the formulas, and have the same dimension as y.) Actually $\underline{y}_{n-1}$ is the vector that would exist at the beginning of step n. The problem is to describe a sequence of operations on $\underline{y}_{n-1}$ that produce $\underline{y}_n$. This description would be an algorithm which embodies the given method.

The first of these operations is the <u>prediction</u> step, which equates $y_{n(0)}$ to a linear combination of the rows of $\underline{y}_{n-1}$:

$$y_{n(0)} = \sum_1^{K_1} \alpha_i\, y_{n-i} \quad + \quad h \sum_1^{K_2} \beta_i\, \dot{y}_{n-i} \ .$$

This is just the formula for an explicit linear multistep method with the same K_1 and K_2 as the given implicit one. There is also a similar formula for predicting $h\dot{y}_n$ from the same data, as

$$h\dot{y}_{n(0)} = \sum_1^{K_1} \gamma_i\, y_{n-i} + h \sum_1^{K_2} \delta_i\, \dot{y}_{n-i}$$

for some coefficients γ_i and δ_i. It turns out that the best choice for these coefficients is

$$\gamma_i = (\alpha_i^* - \alpha_i)/\beta_0^* \ , \qquad \delta_i = (\beta_i^* - \beta_i)/\beta_0^* \ .$$

This choice is arrived at by making $h\dot{y}_{n(0)}$ satisfy the equation

$$y_{n(0)} = \sum_1^{K_1} \alpha_i^*\, y_{n-i} + h\left[\beta_0^*\, \dot{y}_{n(0)} + \sum_1^{K_2} \beta_i^*\, \dot{y}_{n-i}\right],$$

i.e. the original implicit equation with $y_{n(0)}$ in place of y_n and $\dot{y}_{n(0)}$ in place of $\dot{y}_n$. If we use these predicted vectors to construct a predicted value of the entire array $\underline{y}_n$, that value would be

$$\underline{y}_{n(0)} = \left(\begin{array}{cccc|cccc} \alpha_1 & \cdots & & \alpha_{K_1} & \beta_1 & \cdots & & \beta_{K_2} \\ 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & 0 & & & & \\ \hline \gamma_1 & \cdots & & \gamma_{K_1} & \delta_1 & \cdots & & \delta_{K_2} \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & 0 \end{array}\right) \underline{y}_{n-1} = B\,\underline{y}_{n-1} \ .$$

(Compare p. 103 in the text.) All elements not shown are zero. The diagonal lines of 1's have the effect of simply moving the y_{n-i} and $h\dot{y}_{n-i}$ for $i \geq 1$ down one row in the array. The matrix B is L x L.

We now have to describe the corrections to $y_{n(0)}$ to arrive at y_n. Recall from Section 7 (b) that the corrector iteration from $y_{n(m)}$ to $y_{n(m+1)}$ can be written

$$y_{n(m+1)} = y_{n(m)} - P_n^{-1} F(y_{n(m)}),$$

where P_n is a certain matrix and F is a vector function representing the residual, or amount by which the implicit equation is not satisfied. Suppose that we also correct $h\dot{y}_{n(0)}$ at the same time, by making the implicit equation hold for $y_{n(m)}$ and $\dot{y}_{n(m)}$ in place of y_n and $\dot{y}_n$. Then we will have

$$h\dot{y}_{n(m+1)} = h\dot{y}_{n(m)} - \frac{1}{\beta_0^*} P_n^{-1} F(y_{n(m)}).$$

The function F was defined as

$$F(y) = y - \sum_1^{K_1} \alpha_i^* y_{n-i} - h \sum_1^{K_2} \beta_i^* \dot{y}_{n-i} - h\beta_0^* f(y, t_n)$$

so that

$$-\frac{1}{\beta_0^*} F(y_{n(m)}) = hf(y_{n(m)}, t_n) - h\dot{y}_{n(m)} \equiv G(\underline{y}_{n(m)}),$$

where the argument of G is the m^{th} iterate of the entire array being corrected. Thus the correction to $\underline{y}_{n(m)}$ is given by adding $P_n^{-1} G(\underline{y}_{n(m)})$ to $h\dot{y}_{n(m)}$ and $\beta_0^* P_n^{-1} G(\underline{y}_{n(m)})$ to $y_{n(m)}$, and leaving all other rows unchanged. This can be written in the compact form

$$\underline{y}_{n(m+1)} = \underline{y}_{n(m)} + \underline{e} G(\underline{y}_{n(m)}) P_n^{-1}.$$

(The order of the factors G and P^{-1} is reversed here because the vectors $y_{n(m)}$, etc. are row vectors when considered as part of $\underline{y}_n$, while they were column vectors in the discussion in Section 7 (b).) Here $\underline{c}$ is the column vector of length L given by

$$\underline{c} = \begin{pmatrix} \beta_0^* \\ 0 \\ \vdots \\ 0 \\ \hline 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \} K_1 \\ \\ \\ \\ \\ \} K_2 \end{matrix} .$$

P_n^T (transpose matrix) is an appropriate approximation to $I - h\beta_0^* J$, where J is the N x N Jacobian matrix, $\partial f / \partial y$, evaluated at some appropriate point in the neighborhood of (y_n, t_n). Finally, we stop correcting after some number, M, of iterations, and take $\underline{y}_n = \underline{y}_{n(M)}$.

This formulation can also be used in the explicit case, and becomes even simpler. We simply let $P_n = I$, the identity matrix, put $\beta_0^* = 0$, and let the γ_i and δ_i be arbitrary. Then one gets the correct vectors in $\underline{y}_{n(0)}$ already, except in the row for $h\dot{y}_n$. But then the next corrected value in that row is

$$h\dot{y}_{n(1)} = h\dot{y}_{n(0)} + G(\underline{y}_{n(0)}) = hf(y_{n(0)}, t_n) = hf(y_n, t_n),$$

which is the correct value. Thus this scheme gives correct results with M = 1.

These sets of formulas provide a compact description of the algorithm in considerable detail, including the choice of predictor formula, and the choice of corrector iteration method. However, it is not of great value by itself. It becomes much more useful when we consider the use of <u>other</u> history arrays, instead of the natural choice used so far.

As stated earlier, for example, the Adams methods are usually implemented with backward differences of the $\dot{y}$ values, in place of the $\dot{y}$ values themselves. Thus a common history array for the K-step implicit Adams method, for example, is

$$\underline{z}_n = \begin{pmatrix} y_n \\ h\dot{y}_n \\ h\nabla\dot{y}_n \\ \vdots \\ h\nabla^{K-1}\dot{y}_n \end{pmatrix}$$

instead of

$$\underline{y}_n = \begin{pmatrix} y_n \\ h\dot{y}_n \\ h\dot{y}_{n-1} \\ \vdots \\ h\dot{y}_{n+1-K} \end{pmatrix} .$$

Both vectors have L = K + 1 rows. Each of the two sets of data is obtainable from the other by taking linear combinations, as illustrated by the identity

$$\nabla^k \dot{y}_n = \sum_{i=0}^{k} (-1)^i \binom{k}{i} \dot{y}_{n-i} .$$

Thus there is a non-singular L x L matrix T such that

$$\underline{z}_n = T\,\underline{y}_n .$$

With this transformation of history arrays in mind, the algorithm for a step using the $\underline{y}_n$ can be restated as an algorithm for the $\underline{z}_n$. We define new quantities

$$A = TBT^{-1}, \quad \underline{\ell} = T\underline{c}, \quad F(\underline{z}_{n(m)}) = G(\underline{y}_{n(m)}).$$

Then we have

$$\begin{aligned} \underline{z}_{n(0)} &= A\,\underline{z}_{n-1} \\ \underline{z}_{n(m+1)} &= \underline{z}_{n(m)} + \underline{\ell}\,F(\underline{z}_{n(m)})\,P_n^{-1} \\ \underline{z}_n &= \underline{z}_{n(M)} \end{aligned}$$

in place of the analogous relations for $\underline{y}_{n(m)}$. The matrix A and the vector $\underline{\ell}$ for the implicit Adams method of order k = K + 1, as defined by these relations, turn out to be:

$$A = \begin{pmatrix} 1 & \gamma_0 & \cdots & \gamma_{k-2} \\ & 1 & \cdots & 1 \\ & & \cdots & \\ & & & 1 \end{pmatrix}, \qquad \underline{\ell} = \begin{pmatrix} \gamma_{k-1} \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

(Compare p. 148 in the text.)

Another choice for the history array is one invented by Nordsieck. Here we note that the L pieces of data in $\underline{y}_n$ uniquely determine a polynomial $p(t)$ of degree q = L - 1 or less. This polynomial could be written

$$p(t) = \sum_{i=0}^{q} p_i\,(t - t_n)^i .$$

If $N>1$, then $p(t)$ is vector-valued, and the coefficients p_i are vectors of length N. This polynomial is defined by the fact that it interpolates the data in the array $\underline{y}_n$. Moreover, $p(t)$ is generally the polynomial by which the prediction step is done, in that $y_{n+1(0)} = p(t_{n+1})$ if $p(t)$ corresponds to $\underline{y}_n$. Any array which uniquely represents this polynomial is also an alternate representation of the needed history. The idea of Nordsieck was to represent $p(t)$ simply by its coefficients p_i, but multiplied by h^i so as to get quantities which have the same dimension as y and which appear more naturally in the formulas. Thus the Nordsieck history array is

$$\underline{a}_n = \begin{pmatrix} p_0 \\ p_1 h \\ \vdots \\ p_q h^q \end{pmatrix} = \begin{pmatrix} p(t_n) \\ h\dot{p}(t_n)/1! \\ \vdots \\ h^q p^{(q)}(t_n)/q! \end{pmatrix}.$$

The rows of this array then approximate the quantities $h^i y^{(i)}(t_n)/i!$, $0 \le i \le q$, but only up to order q, i.e. within $O(h^{q+1})$.

One of the main reasons for this choice of history array is that it makes it relatively easy to change the step size h. For if $\underline{a}_n$ exists and the step to t_{n+1} is to be taken with a step size h', then the rows of $\underline{a}_n$ are simply rescaled by the powers $(h'/h)^i$, $0 \le i \le q$, before taking that step.

Another reason is that the estimation of local errors is relatively easy from $\underline{a}_n$, because that requires estimates of derivatives of y. A third reason is that it is very easy to interpolate to get a computed estimate of y(t) for any intermediate t, $t_{n-1} < t < t_n$. This is because p(t) is given as an obvious linear combination of the rows of $\underline{a}_n$, by writing a Taylor series to q+1 terms about t_n.

If a given method is put into a form that uses the Nordsieck array $\underline{a}_n$ for the history, the matrix $A = TBT^{-1}$ needed to do the prediction turns out to be simply

$$A = \begin{pmatrix} 1 & & & \\ & \binom{j}{i} & & \\ 0 & & & 1 \end{pmatrix} = \text{Pascal Triangle},$$

regardless of the original method. This is because the Taylor series for $h^i y^{(i)}(t_n)/i!$ about t_{n-1} begins with the terms

$$\sum_{j=0}^{q} \binom{j}{i} h^j y^{(j)}(t_{n-1})/j! \quad .$$

Moreover, the product Ax of any vector x and the Pascal triangle matrix A can be computed with no multiply operations at all, only additions. Thus prediction is considerably easier with the Nordsieck array than with most other formulations.

The Fortran program given in the text (Chapter 9) is an implementation of the implicit Adams methods and also of Gear's methods, with the use of the Nordsieck history array. It uses the techniques of Section 7 (b) for solution

of the implicit equation and the techniques of Section 7 (c) to adjust the step size and order in relation to a local error tolerance.

If a linear multistep method uses a history vector of length L, i.e. an array of size L x N for a system of N equations, then it is called an L-value method. If L is not specified, it is called a multivalue method. This name, as distinguished from the name multistep method, emphasizes the fact that it is the number of values of data in the history vector that is the more significant, not the number of steps. In fact, the class of multivalue methods, including especially ones that use the Nordsieck history array, is properly larger than the class of multistep methods.

10. Extrapolation methods*

The class of methods now referred to as extrapolation methods was first introduced formally by Richardson in 1927. The idea was then called "deferred approach to the limit," and has also been referred to as Richardson extrapolation.

The following simple example illustrates the basic idea of extrapolation methods. Consider the ODE

$$dy/dt = \lambda y , \quad 0 \le t \le 1$$

with $y(0) = y_0$ given. Suppose we use the (explicit) Euler method on this problem, with various values of Δt. The method, for this problem, is given by

$$y_{n+1} = y_n + \Delta t\, \lambda y_n = (1 + \lambda \Delta t) y_n .$$

For simplicity, take $y(0) = y_0 = 1$, so that the true solution is $y(t) = e^{\lambda t}$. If we use $\Delta t = 1$ for one step, we get the calculated value $y_1 = 1 + \lambda$. If we use $\Delta t = 1/2$ and take two steps, the first yields $y_1 = 1 + \lambda$, and the second $y_2 = (1 + \lambda)\, y_1 = (1 + \lambda)^2$ as the computed value for $y(1)$. Denote by T_0^k ($k = 0, 1, \ldots$) the values obtained for $\Delta t = 1, \frac{1}{2}, \frac{1}{4}, \ldots$ at $t = 1$. We then arrive at the following values

$$\Delta t = 1: \quad T_0^0 = 1 + \lambda$$

$$\Delta t = 1/2: \quad T_0^1 = (1 + \lambda/2)^2 = 1 + \lambda + \lambda^2/4$$

$$\Delta t = 1/4: \quad T_0^2 = (1 + \lambda/4)^4 = 1 + \lambda + \tfrac{3}{8}\lambda^2 + \tfrac{1}{16}\lambda^3 + \tfrac{1}{256}\lambda^4 .$$

---

*The lecture covering this chapter was given by Julius Chang. The notes for it were extracted from the report, "An Introduction to Extrapolation Methods for the Numerical Solution of Differential Equations," UCID-15992, Feb. 1972, by Julius Chang.

The idea now is to use these three values to arrive at an approximation to y(1) that is more accurate than any of them, by taking certain linear combinations. Specifically, construct from the above three T_0^k the two quantities

$$T_1^0 = \frac{2T_0^1 - T_0^0}{2-1} = 1 + \lambda + \lambda^2/2$$

$$T_1^1 = \frac{2T_0^2 - T_0^1}{2-1} = 1 + \lambda + \lambda^2/2 + \lambda^3/8 + \lambda^4/128.$$

(The justification for these formulas will be seen later.) Finally, from these two T_1^k, construct

$$T_2^0 = \frac{4T_1^1 - T_1^0}{4-1} = 1 + \lambda + \lambda^2/2 + \lambda^3/6 + \lambda^2/96.$$

Compare these to the true value

$$y(1) = e^{\lambda} = 1 + \lambda + \lambda^2/2 + \lambda^3/6 + \lambda^4/24 + \cdots.$$

We see that the T_0^k are accurate only to the λ term, consistent with the fact that the Euler method is of order 1. But the T_1^k are accurate to the λ^2 term, and T_2^0 is accurate to the λ^3 term.

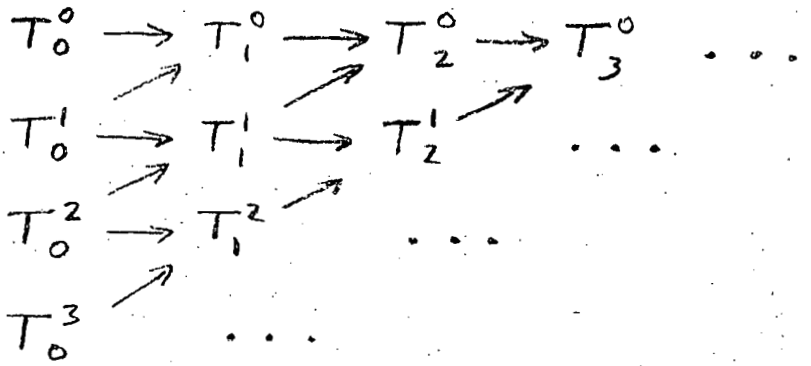
Notice that the quantities having greater accuracy than those given by the basic method were obtained simply by taking appropriate linear combinations of the latter. It was not necessary to refer again to the differential equation or to any higher order method. This is significant when we consider the ODE here as having the general form dy/dt = f(y,t), and realize that in practical problems, the cost of computing values of f is the dominant cost factor of any given method. The cost of performing arithmetic operations of the type shown above for the T_m^k is usually negligible by comparison. In the present example, f was computed only two times to obtain the values T_0^0 and T_0^1 and hence T_1^0, namely for $\dot{y}_0 = f(y_0, 0)$ and $f(y_0 + \frac{1}{2}\dot{y}_0, \frac{1}{2})$. Then

f was computed 3 more times to get T_0^2 and hence T_1^1 and T_2^0. If we were to perform the Euler method for the value of Δt which requires the same number of f evaluations, namely $\Delta t = \frac{1}{5}$, the resulting approximation to y(1) would be

$$\Delta t = \tfrac{1}{5}: \quad y_5 = \left(1 + \tfrac{\lambda}{5}\right)^5 = 1 + \lambda + \tfrac{2}{5}\lambda^2 + \cdots .$$

This is clearly much less accurate than T_2^0, while the cost of both quantities (in terms of f evaluations) is the same.

The algorithm can be easily extended to any desired extent. We write an array of T's, as follows:



The arrows show the dependence of each T on the ones to the left of it. The result is a triangular array, which is carried to whatever extent is needed to produce results of acceptable accuracy.

The algorithms for generating these approximations, which will be given in more detail shortly, depend on certain theoretical properties of the basic method (the Euler method in the above example). This theory ignores the effect of machine roundoff error, which can cause some difficulty for extrapolation methods, whereas it rarely does for other classes of methods on modern computers.

The fundamental assumption made here is that if y_n is the calculated approximation to $y(t_n)$, using a given method with step size Δt and taking steps until $t_n = t$ (a fixed value), then

$$y_n = y(t_n) + (\Delta t)^{p_1} \epsilon_1(t_n) + (\Delta t)^{p_2} \epsilon_2(t_n) + \cdots + (\Delta t)^{p_k} \epsilon_k(t_n) + O((\Delta t)^{p_{k+1}}),$$

where the p_i are integers with $1 \le p_1 < p_2 < \ldots$, and the $\epsilon_i(t)$ are continous functions which do not depend on Δt. The p_i and ϵ_i will of course depend on the basic method being used. The expansion represents explicitly the error in y_n relative to the true value $y(t_n)$.

Now, for a fixed value of $t = t_n$, we consider the use of the method with several values of Δt, say Δt_0, Δt_1, ..., each a submultiple of t (so that for each Δt_i, $t_n = t$ after some number n of steps). We write down the resulting approximations of $y(t)$ and their expansions according to the above formula:

$$\Delta t = \Delta t_0: \quad y_n^0 = y(t) + (\Delta t_0)^{p_1} \epsilon_1(t) + \cdots$$
$$\Delta t = \Delta t_1: \quad y_n^1 = y(t) + (\Delta t_1)^{p_1} \epsilon_1(t) + \cdots$$
$$\cdots$$

After building up $k + 1$ of such values, y_n^0, y_n^1, ..., y_n^k (the n's are different for the different Δt values), we want to use their expansions to eliminate the unknown error terms, and get a better approximation to $y(t)$ as a result.

Multiply the equation for y_n^0 by some undetermined coefficient a_0, the next equation by a_1, and so on, until multiplying the equation for y_n^k by a_k, and then add up the results. The sum is

$$\bar{y} = \sum_0^k a_i y_n^i = (\Sigma a_i)\, y(t) + (\Sigma a_i (\Delta t_i)^{p_1}) \epsilon_1(t) + (\Sigma a_i (\Delta t_i)^{p_2}) \epsilon_2(t) + \cdots .$$

To eliminate the error terms, and get y(t) as a result, we want the a_i to satisfy

$$\Sigma a_i = 1$$

$$\Sigma a_i (\Delta t_i)^{p_m} = 0 , \quad m = 1, 2, \ldots, k .$$

This would make

$$\bar{y} = y(t) + O((\Delta t_0)^{p_{k+1}}).$$

The conditions on the a_i form a linear system of k + 1 equations in k + 1 unknowns. The matrix for that linear system is of van der Monde type, and so is not singular. So we can, in theory, compute the solution and hence $\bar{y}$, which has a high order of accuracy.

The trouble with this approach is that the linear system for the a_i can be very "ill-conditioned," meaning that roundoff errors are likely to build up to an intolerable level during the computation. For this reason, the actual algorithm recommended is not as indicated above, but instead involves formulas for the direct calculation of quantities of the type $\bar{y}$, as done in the example earlier.

This algorithmic approach can be obtained by taking an alternate point of view on the expansion for y_n for a given Δt. If we ignore the last term, $O((\Delta t)^{p_{k+1}})$, then that expansion gives an approximation to y_n as a polynomial in Δt, $p(\Delta t)$, of degree p_k. This polynomial has coefficients $\epsilon_i(t)$ and a constant term of $p(0) = y(t)$. Thus the value $p(0)$ is what is sought, but what is actually known is a collection of values of $p(\Delta t)$ for nonzero Δt, by way of the calculations of y_n by the basic method. This

viewpoint reduces the problem to that of fitting a polynomial of a specified form to a given set of data. That class of problems has been studied in its own right quite thoroughly, and algorithms for such polynomial fitting problems are common. As applied in the context of ODE's, this idea leads to the polynomial extrapolation methods, as studied by Gragg (Ref. 19).

Another variation of these ideas is the use of rational functions. Here y_n is approximated by a rational function of Δt, accurate to some high order, and that function is fitted by the use of an appropriate number of calculated values of y_n with different Δt. Then the fitted rational function is evaluated at $\Delta t = 0$ to give a highly accurate approximation to $y(t)$. Such methods are rational extrapolation methods, and have been studied by Bulirsch and Stoer (Ref. 20).

The rational extrapolation methods are still being actively developed. But it appears that they are quite competitive for certain large classes of problems. The extrapolation methods of both types have the desirable feature that the order of accuracy obtained is variable, depending on the accuracy needs of the problem. That is, the extent, as measured by the index k above, to which extrapolation is carried can vary from step to step, as controlled by an appropriate error tolerance criterion.

For polynomial extrapolation, the details of the algorithm for constructing the T array are as follows. The triangular array

$$
\begin{array}{cccc}
T_0^0 & & & \\
T_0^1 & T_1^0 & & \\
T_0^2 & T_1^1 & T_2^0 & \\
T_0^3 & T_1^2 & T_2^1 & T_3^0 \\
\cdots & & &
\end{array}
$$

is generated, either by rows or by columns, with a procedure shown schematically by

$$\begin{array}{ccc} T_m^k & & \\ & \searrow & \\ T_m^{k+1} & \rightarrow & T_{m+1}^k \end{array} .$$

In the case that $p_i = i$, which is the case for the Euler method, the formula for this procedure is

$$T_{m+1}^k = \frac{(\Delta t_k / \Delta t_{k+m+1}) T_m^{k+1} - T_m^k}{(\Delta t_k / \Delta t_{k+m+1}) - 1} .$$

Here Δt_k is the value of Δt used to generate $y_n = T_0^k$ by the basic method. If a basic method of order 2 is used, and we have $p_i = 2i$, we use instead the formula

$$T_{m+1}^k = \frac{(\Delta t_k / \Delta t_{k+m+1})^2 T_m^{k+1} - T_m^k}{(\Delta t_k / \Delta t_{k+m+1})^2 - 1} .$$

In either case, the formula is applied with $k \geq 0$, $m \geq 0$, $k + m \leq \bar{m}$, where $\bar{m}$ is the number of rows or columns to be generated. Then $T_{\bar{m}}^0$ is the final approximation to $y(t)$.

The above formulas are well-behaved with respect to roundoff error, in that the ratios $\Delta t_k / \Delta t_{k+m+1}$ are usually considerably larger than 1. In fact, the condition on the Δt values under which convergence of the method can be proved theoretically is that

$$\sup_m \Delta t_{m+1} / \Delta t_m < 1 .$$

This is equivalent to a statement that the ratios $\Delta t_m / \Delta t_{m+1}$ are larger than 1 and are bounded away from 1.

For rational extrapolation, the details of the algorithm are somewhat more complicated. (But the quality of the results makes up for this.) Here the only case for which the algorithm has been developed is that in which only even powers of Δt appear in the basic expansion, i.e. $p_i = 2i$. Then the rational function used to fit that expansion is the quotient of two polynomials in $(\Delta t)^2$, so that it also involves only even powers of Δt. The T array has the following form:

$$
\begin{array}{ccccccc}
T_0^0 & & & & & & \\
T_{-1}^1 & & T_1^0 & & & & \\
T_0^1 & & T_2^0 & & & & \\
T_{-1}^2 & & T_1^1 & & T_3^0 & & \\
T_0^2 & & T_2^1 & & & & \\
T_{-1}^3 & & T_1^2 & & \cdots & & \\
T_0^3 & & \cdots & & & &
\end{array}
$$

The array requires a leftmost column of values $T_{-1}^k = 0$ in order to get the recurrence formula started. That formula is

$$
T_{m+1}^k = T_m^{k+1} + \frac{T_m^{k+1} - T_m^k}{\left(\frac{\Delta t_k}{\Delta t_{k+m+1}}\right)\left[1 - \frac{T_m^{k+1} - T_m^k}{T_m^{k+1} - T_{m-1}^{k+1}}\right] - 1}.
$$

We come now to the choice of the basic method, to which extrapolation is to be applied. The example given at the beginning of the chapter involved the Euler method,

$$y_{n+1} = y_n + \Delta t\, f(y_n, t_n),$$

as the basic method. For this choice we have $p_i = i$ in the expansion formula. A more powerful choice would be to take a second order method, such as the trapezoid rule,

$$y_{n+1} = y_n + \frac{\Delta t}{2}\left[f(y_{n+1}, t_{n+1}) + f(y_n, t_n)\right].$$

Here $p_i = 2i$, so that the above rational extrapolation algorithm could be applied. However, this is an implicit method, and the solution of the implicit equation will inevitably involve some iteration error. Such error is not accounted for in the theory behind the basic expansion formula, and its presence is likely to reduce the accuracy of the extrapolated values.

A third choice, which is the one adopted by Bulirsch and Stoer, is the modified midpoint rule. Here the basic step from $t = t_n$ to $t = t_{n+1} = t_n + \Delta t$ is given as follows: Let $M \geq 1$, $\Delta\tau = \Delta t/2M$, and $\tau_m = t_n + m\Delta\tau$ for $m=0, 2, \ldots, 2M$ ($\tau_{2M} = t_{n+1}$). Then let

$$\begin{aligned} z_0 &= y_n \\ z_1 &= z_0 + \Delta\tau\, f(z_0, \tau_0) \\ z_{m+1} &= z_{m-1} + 2\Delta\tau\, f(z_m, \tau_m), \quad m = 1, 2, \ldots, 2M-1, \\ y_{n+1} &= \tfrac{1}{2}\left[z_{2M} + z_{2M-1} + \Delta\tau\, f(z_{2M}, \tau_{2M})\right]. \end{aligned}$$

In other words, the interval of size Δt is broken into $2M$ equal parts. A step is taken with the Euler method, followed by $2M-1$ steps with the midpoint

rule, and a final smoothing operation is performed at the end to get y_{n+1}. The final smoothing adds stability to the method. This method also has an expansion formula with even powers of Δt, so that $p_i = 2i$. This means that each extrapolation stage, which eliminates one term in that expansion, reduces the error in the computed value by $(\Delta t)^2$, rather than Δt, as is the case when $p_i = 1$.

There are various theorems concerning the convergence properties of these extrapolation methods. These can be found in the book by Lapidus and Seinfeld, (Ref. 4). The main conclusion of these is that the diagonal element of the T array, T_m^o, converges rapidly, under appropriate conditions, to the true solution. This conclusion is weakened in practice only by the presence of roundoff error. Such error can be quite harmful when the Euler method is the basic method, but for the others does not seem to be so harmful.

The storage necessary to perform extrapolation methods is not as bad as it first appears. The T array, if constructed in a natural order, need not be stored in its entirety. Only one or two rows of T at a time are needed, the others being discarded after they are used. Thus only a vector or two of values is stored, not a matrix.

The extrapolation methods have been used locally for various calculations, and have led to varying degrees of success. One area in which they have been tried is the solution of partial differential equations, where a discretization in the space variables leads to a system of ODE's. The use of rational extrapolation on these ODE systems has been quite successful.

Part III: Stiff Problems

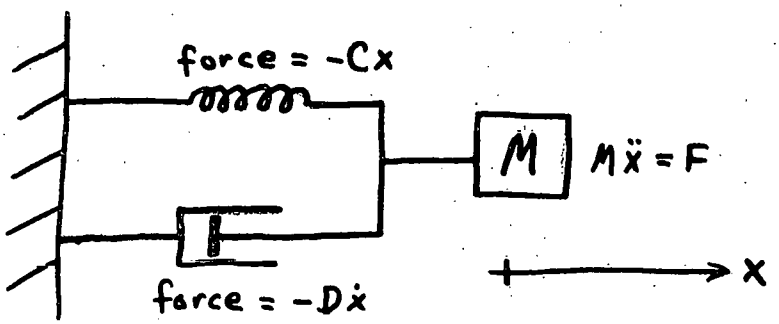
11. Stiffness

Two examples of stiff problems have been given so far, namely

$$\dot{y} = -1000\,(y - t^2) + 2t\,, \quad y(0) = 0\,,$$
$$\dot{y} = -100\,(y - e^{-t}) - e^{-t}, \quad y(0) = 1\,.$$

In both cases, the large negative factor on the y term of f(y,t) makes some ODE methods take much longer to solve these problems than would be expected from their simple and well-behaved solutions, namely $y = t^2$ and $y = e^{-t}$.

Both of these are contrived problems, designed to illustrate a difficulty that can occur in the numerical solution of an ODE. But that difficulty itself is not contrived. To show that it is not, the following is a realistic mechanical example problem. Consider an object of mass M attached to a fixed wall by a spring and a dashpot, and free to move along the x axis (gravity is ignored). The spring exerts a resistive force of -Cx on the object, where C is the Hooke's Law constant. The dashpot also exerts a resistive force, but in proportion to velocity: the force is $-D\dot{x}$ where D is a constant. The motion obeys Newton's Second Law,



$$M\ddot{x} = \text{total force} = -Cx - D\dot{x}\,.$$

By making simple changes of variables, we can effect the normalization M = C = 1/2. The ODE is then

$$\ddot{x} + 2D\dot{x} + x = 0\,.$$

The initial position x_0 and initial speed $\dot{x}_0$ would be specified for the initial value problem.

The solution of this problem is a linear combination of two single exponentials:

$$x = Ae^{\lambda_1 t} + Be^{\lambda_2 t},$$

where the λ_j are the roots of the quadratic

$$\lambda^2 + 2D\lambda + 1 = 0.$$

(They are also the eigenvalues of the 2 x 2 matrix J if the ODE is written as a system $\dot{y} = Jy$ with N = 2.) Thus

$$\lambda_1, \lambda_2 = -D \pm \sqrt{D^2 - 1}.$$

Now consider the case in which the dashpot is very <u>stiff</u> (mechanically speaking), in that D is very large compared to 1: $D \gg 1$. In that case, the above roots can be approximated as

$$\lambda_1 \approx -D - \sqrt{D^2} = -2D,$$

$$\lambda_2 = -D + \sqrt{D^2 - 1} = \frac{(-D + \sqrt{D^2-1})(-D - \sqrt{D^2-1})}{-D - \sqrt{D^2-1}}$$

$$= \frac{-1}{D + \sqrt{D^2-1}} \approx -1/2D.$$

Thus $\lambda_1 \ll 0$, and $e^{\lambda_1 t}$ is a very rapidly decaying term, while $\lambda_2 \approx 0$, and $e^{\lambda_2 t}$ is very slowly decaying. We would say that the rapid decay process corresponds to a very short <u>time constant</u>

$$\tau_1 = -1/\lambda_1 \approx 1/2D,$$

while the slow process corresponds to a long time constant

$$\tau_2 = -1/\lambda_2 \approx 2D.$$

In the numerical solution of this problem, by methods such as the higher order Adams methods which are not designed for stiff problems, the

short time constant τ_1 will have the effect of controlling the time step. That is, stable numerical behavior will not occur unless the step size h is comparable to τ_1. But the physical behavior of the solution, at least after a short initial time period, is governed by the long time constant τ_2. Thus to the extent that $\tau_2/\tau_1 = 4D^2$ is larger than 1, the step sizes will be much smaller than what is physically reasonable. This typifies the difficulty of stiffness.

A definition of stiffness suitable for most purposes can be given for the general problem $\dot{y} = f(y,t)$. We consider the N x N Jacobian matrix $J = \partial f/\partial y$, and its N eigenvalues λ_j. We suppose that all the $Re(\lambda_j) < 0$, so that the exponentials $e^{\lambda_j t}$ are decaying. Define real numbers $\tau_j > 0$ and ω_j by

$$\lambda_j = -\frac{1}{\tau_j} + i\,\omega_j .$$

Then in a local sense, the solution will behave approximately like a linear combination of the fundamental solutions

$$e^{\lambda_j t} = e^{-t/\tau_j}\, e^{i\omega_j t} .$$

The first factor is a damping factor, and the second is an oscillatory factor. The positive numbers $\tau_j = -1/Re(\lambda_j)$ are the time constants of the system. The damping factor in $e^{\lambda_j t}$ decays by a factor of 1/e in a time of τ_j.

Definition: An ODE system with positive time constants τ_j is called stiff if

$$\max \tau_j \,/\, \min \tau_j \gg 1 .$$

The quantity $\tau_{max} = \max \tau_j$, the largest of the time constants, represents the rate of decay of the most slowly decaying of the fundamental solutions

and therefore indicates the time span of interest for the problem. Similarly, $\tau_{min} = \min \tau_j$ represents the rate of decay of the most rapidly decaying term. The other time constants are in between these extremes. The ratio τ_{max}/τ_{min} describes the relative spread in the time constants, and is called the stiffness ratio.

In the two single example equations given at the beginning of the chapter, there is no matrix, only the scalar quantity $\lambda_1 = \partial f/\partial y$, which is -1000 or -100 there. Thus only the time constant $\tau_1 = -1/\lambda_1$ (=.001 or .01) is explicitly present. However, in those equations, the true solutions (t^2 and e^{-t}), which are smoothly varying over, say, $0 \leq t \leq 1$, represent inherent time constants on the order of 1. That is, the true solutions are such that step sizes that are not much smaller than 1 would seem reasonable. Hence, since $\tau_1 \ll 1$, these problems have the qualitative property of stiffness, despite the fact that they do not fit into the above analytic definition.

Another situation not covered by the definition is where there is a possibility of growing fundamental solutions. That is, a problem may have some λ_j with $Re(\lambda_j) > 0$, and yet be considered stiff because of other λ_j with $Re(\lambda_j) \ll 0$.

What is common to all stiff problems is the presence of at least one eigenvalue λ of $\partial f/\partial y$ for which $Re(\lambda) \ll 0$. The questions of how negative that real part must be to call the problem stiff, and what it shoud be compared to in order to give a quantitative measure of stiffness, are often difficult to answer with any precision.

It should also be emphasized that the eigenvalues λ_j in the definition above are as computed from J at some given point (y,t). In general, the value of J changes along the solution curve, and hence so do the λ_j. Thus stiffness is only a <u>local</u> property.

To look at the effect of stiffness on a numerical method, we examine the numerical solution given by a linear multistep method. Locally, that numerical solution will behave approximately like the linear combination

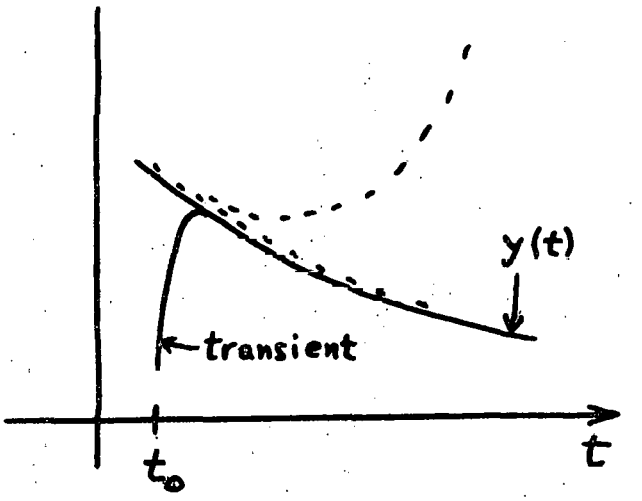
$$\sum_{i=1}^{N} \sum_{j=1}^{K} c_{ij} \left[\xi_j (h\lambda_i) \right]^n$$

in terms of the roots $\xi_j(h\lambda)$ of $\rho + h\lambda\sigma$. (A derivation of this was given for the single test equation $\dot{y} = \lambda y$ in Section 9 (e). For the more general case, the usual linear approximation and diagonalization procedure gives the result, at least in the case of a smoothly varying Jacobian which is diagonalizable, and assuming $\rho + h\lambda\sigma$ has no multiple roots.) In view of the above, and the fact that the true solution is decaying when all $Re(\lambda_j) < 0$, we would want to have $|\xi_j(h\lambda_i)| \leq 1$ for all i and j in order that the numerical solution also decay (or at least not grow exponentially). That is, we want $h\lambda_i \in S_a$ for all i, where S_a is the absolute stability region, in order to maintain a stable numerical behavior.

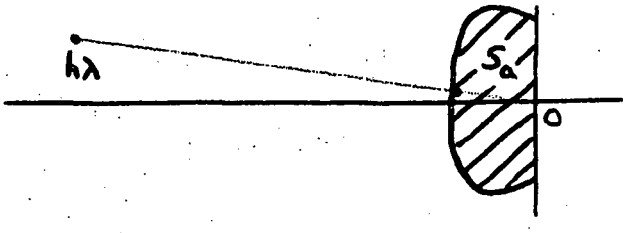
While the above discussion is in the context of linear multistep methods, the same applies to Runge-Kutta and other methods as well. For any given method, with an absolute stability region S_a associated with it, the values of h for which the numerical solution of $\dot{y} = \lambda y$ behaves in an absolutely

stable manner must satisfy $h\lambda \in S_a$, by definition. Then, by the same approximation argument as used before, we infer that for the general problem $\dot{y} = f(y,t)$, in which the eigenvalues λ_i of the Jacobian have $Re(\lambda_i) < 0$, stable numerical behavior will be maintained only if all $h\lambda_i \in S_a$. The details of this latter step in the argument depend on the particular method in question.

The distinction between stable and unstable numerical behavior can be shown graphically for a single equation. The true solution curve behaves approximately like the slowly decaying exponential $e^{-t/\tau_{max}}$ (oscillatory behavior ignored), except for a short initial transient period in which the dominant term involves $e^{-t/\tau_{min}}$. A stable numerical method will generate points that follow the true solution curve well, at least qualititatively. But an unstable one, in which $h\lambda_i \notin S_a$ for some i, will generate points that diverge from the true solution and diverge in an unbounded manner as n grows.



When the problem is stiff, the difficulty imposed by the requirement that all $h\lambda_i \in S_a$ is clear if S_a is a <u>finite</u> region. For those eigenvalues λ_i with highly negative real



parts, h must be made very small in order to force $h\lambda_i$ into S_a.

An alternate way of stating the difficulty is to refer again to the time constants τ_j. In order to complete the numerical solution of the problem efficiently, a reasonable value of the step size h would be, say, between $\tau_{max}/10$ and τ_{max}, since this largest time constant governs the long-term behavior of the solution. But suppose, for example, that S_a extends only as far as $Re(h\lambda) = -5$ in the leftward direction. Then the stability requirement that all $h\lambda_i \in S_a$ implies in particular that $Re(-h/\tau_{min}) \geq -5$, or $h \leq 5\tau_{min}$. Then if $\tau_{max}/\tau_{min} = 1000$, the largest value of h allowed by absolute stability is smaller than the smallest value allowed by efficiency considerations by a factor of $1000/5 \cdot 10 = 20$. Thus for such a method, at least 20 times as many steps will be necessary as would seem reasonable on the basis of the actual behavior of the solution. If the stiffness ratio is larger, the discrepancy between the two conflicting demands on h will be larger, according to that ratio. Stiffness ratios as large as 10^9 are not uncommon.

An important but difficult problem for the user of ODE methods is that of determining whether a given problem is stiff or not. It is not generally feasible to compute the Jacobian J and its eigenvalues for this purpose, both because of the expense of this computation and because of the uncertainty as to what values of (y,t) at which to compute J. (The situation at the initial point (y_0, t_0) may not be at all indicative of the situation later.)

One way to infer that a problem is stiff is to use physical intuition in relation to the system being modeled or described by the ODE. If there is reason to believe that that system includes one or more very rapidly decaying processes along with slower ones, then the ODE is likely to be stiff. Unfortunately, the converse statement is frequently false: There may be no such rapid decay processes inherent in the system on physical grounds, but yet the ODE is still stiff. This can usually be attributed to the way in which the ODE is constructed as a mathematical model of the physical system, such as in the discretization process of converting a partial differential equation to an ODE system. When stiffness is not apparent from physical or other similar considerations, the only recourse is experimental computation on the problem. That is, if a solution is attempted with a method that is unsuitable for stiff problems, and the calculation requires much smaller steps that is reasonable for the solution behavior, it is quite likely that the problem is stiff. In that case, of course, a method that <u>is</u> suitable for stiff problems should run much more efficiently on it.

12. Methods for stiff problems

12. (a) A-Stability; Dahlquist's Theorem

From the graphical illustration given above of the dilemma imposed by a finite absolute stability region S_a on a stiff problem, it should be clear how one might look for methods which are suitable for such problems. Namely, methods for which S_a is not finite should be sought. More specifically, a desirable method for the solution of stiff problems is one for which S_a extends infinitely far to the left in the $h\lambda$ plane. This would allow values of $h\lambda$ to be in S_a despite the large negative value of $\mathrm{Re}(h\lambda)$.

One simple way to satisfy this need, and the way that first arose historically, is to select methods which are A-stable, i.e. which include the entire left half-plane $\{h\lambda : \mathrm{Re}(h\lambda) \leq 0\}$ in S_a. For such methods, the stability condition that $h\lambda \in S_a$ for all eigenvalues λ having $\mathrm{Re}(\lambda) < 0$ is automatically fulfilled, and the step size is restricted only by accuracy considerations.

We have already seen two A-stable methods. One is the implicit Euler method (Chapter 4), and the other is the trapezoid rule (Exercise 6.1). These are also the implicit Adams methods of orders 1 and 2, respectively. However, among the linear multistep methods, there are very few other choices of A-stable methods available. The reason for this is given in the following theorem, the rather famous theorem of Dahlquist:

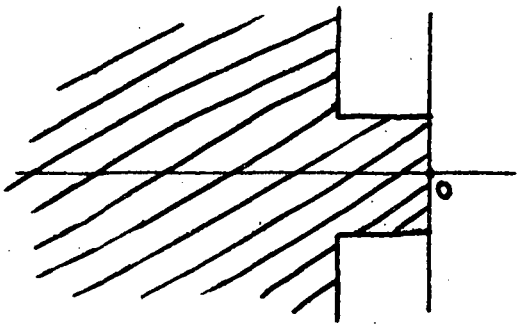
Theorem (Dahlquist): An A-stable linear multistep method has order $r \leq 2$. Of those A-stable methods of order 2, the one with smallest local truncation error coefficient is the trapezoid rule.

This says that within the class of linear multistep methods, the search for A-stable methods leads essentially only to the implicit Euler method and the trapezoid rule, these being the natural choices for orders 1 and 2. Outside of that class, the A-stable methods include some of the implicit Runge-Kutta methods (see Section 8 (d)), some composite linear multistep methods (mentioned in Section 5 (d)), and a few others developed recently. Unfortunately, all of these methods have certain drawbacks. At the present state of the subject, they all seem to lack the efficiency that is attained with the better methods for non-stiff problems. This is due to either the restriction to low order, or the problem of solving large systems of implicit equations, or other difficulties with the implementation of the methods.

12. (b) Stiff stability

The condition of A-stability, as a requirement for a method to be considered for stiff problems, leads to disappointingly few methods. A more fruitful approach to the subject was followed by Gear. In this approach, instead of requiring that the entire half-place $\mathrm{Re}(h\lambda) \leq 0$ lie in the absolute stability region S_a, it is only required that a suitably

large part of it lie in S_a. Specifically, Gear defines a method to be stiffly stable if its S_a includes a region consisting of a rectangle to the left of the origin together with a half-plane adjoining that to the left (as shown in figure).



Thus for a stiffly stable method, values of $h\lambda$ with $Re(\lambda) \ll 0$ will still belong to S_a, no matter how large $|Re(\lambda)|$ gets. The only values of λ which can cause an instability here, but which do not for the A-stable methods, are those with sizable values of $Im(\lambda)$ in relation to $Re(\lambda)$, depending on the particular shape of S_a in the above figure. This is a relatively small price to pay for the obtaining of methods that treat stiff problems efficiently.

If one now searches the class of linear multistep methods for stiffly stable methods, there are quite a few choices available. All of these are <u>implicit</u>, because the region S_a for any explicit method can easily be shown to be bounded. Among these choices are the methods of Gear (see Section 9 (c)), which are discussed in the next section.

(The explicit Runge-Kutta methods are also excluded here, because of their finite S_a.)

For any of the stiffly stable linear multistep methods, the stability problem is taken care of by the nature of S_a, in the presence of stiffness. Of course, the method must be examined for consistency, order of accuracy, and stability in the limit of small $h\lambda$ (where strong stability is preferred).

Another important condition for such a method to be useful is that the implicit equation be solvable. For if that equation is not solved fairly accurately, the desirable stability properties, which are derived for the exact solution of the implicit formula, will be lost.

Suppose we contemplate the use of functional iteration for the solution of the implicit equation. Recall from Section 7 (b) that this iterative method will in general converge only when $|\beta_0|hL < 1$, where L is the Lipschitz constant for f. But if the problem is stiff, L will be of about the same size as $\max|\lambda_j|$, the largest eigenvalue in magnitude, and so will be about $1/\tau_{min}$ or greater. Since $|\beta_0|$ is generally not much smaller than 1, this convergence condition reduces to, approximately, the condition $h < \tau_{min}$. This is just the kind of restriction on step size that we are attempting to avoid by selection of stiffly stable methods. We must therefore conclude that functional iteration is not an acceptable method for solving the implicit equation in the case of a stiff problem.

This leaves the various iteration methods based on Newton's method, i.e. the quasi-Newton on chord methods, as the logical choice for treating the implicit equation. The chord methods require the solution of a linear system at each iteration. This introduces a significant overhead cost in the implementation of the method, as compared to functional iteration. But for the general stiff problem there is no choice but to put up with the added overhead in such a manner. The alternative would be an increase in the

number of steps, and thus in the number of evaluations of f, that is intolerable.

12. (c) Gear's methods

In this section, we look more closely at the methods of Gear. These are linear multistep methods of the form

$$y_n = \sum_{i=1}^{K} \alpha_i y_{n-i} + h\beta_0 \dot{y}_n .$$

In terms of the general linear multistep formula, note that $K_1 = K$, $K_2 = 0$, and that only the forwardmost value of $\dot{y}$, $\dot{y}_n = f(y_n, t_n)$, appears. This is a K-step method, and is clearly implicit. There are K + 1 coefficients involved, and this is sufficient to achieve order K, which is what is done.

These methods can be viewed as formulas for numerical differentiation, by writing $\alpha_0 = -1$ and

$$h\dot{y}_n = -\sum_{i=0}^{K} \frac{\alpha_i}{\beta_0} y_{n-i} .$$

In this form, they are known as the Backward Differentiation Formulas (BDF) of various orders $K = 1, 2, \ldots$. They are most easily derived by writing the operator equation

$$hD = \log E = \log\left(\frac{1}{1-\nabla}\right) = \nabla + \frac{\nabla^2}{2} + \frac{\nabla^3}{3} + \cdots$$

in terms of the operators D, E, and ∇ given by

$$Dy = \dot{y}\ , \quad Ey(t) = y(t+h), \quad \nabla y(t) = y(t) - y(t-h)$$

(see Section 9 (c)). By applying hD to $y(t_n)$ we have the infinite series formula

$$h\dot{y}(t_n) = \left(\nabla + \frac{\nabla^2}{2} + \cdots\right) y(t_n) .$$

By truncating this series to K terms, and writing y_n for $y(t_n)$ and $\dot{y}_n$ for $\dot{y}(t_n)$, we obtain

$$h\dot{y}_n = \left(\nabla + \frac{\nabla^2}{2} + \cdots \frac{\nabla^K}{K}\right) y_n = \sum_1^K \frac{1}{j} \nabla^j y_n .$$

This is a restatement of the K^{th} order Gear formula in terms of backward differences, since the $\nabla^j y_n$ are linear combinations of y_n, y_{n-1}, ..., y_{n-K}.

As viewed in the above terms, the local truncation error in the formula can be approximated by the first neglected term,

$$\frac{\nabla^{K+1} y_n}{K+1}$$

Since ∇ is approximated by hD to first order, $\nabla^{K+1} y_n$ is approximated by $h^{K+1} y^{(K+1)}(t_n)$ within terms of higher order in h. If we then multiply through by β_0 to put the formula in its standard form, then the local truncation error, defined as

$$\sum_0^K \alpha_i\, y(t_{n-i}) + h\beta_0\, \dot{y}(t_n),$$

can be approximated as

$$\frac{\beta_0}{K+1} \nabla^{K+1} y_n \approx \frac{\beta_0 h^{K+1}}{K+1} y_n^{(K+1)}$$

within terms that are $O(h^{K+1})$. This again shows that the order of the method is K.

The value of β_0 can be seen by comparing the two different formulations of the same method:

$$h\dot{y}_n = \sum_1^K \frac{1}{j} \nabla^j y_n = -\sum_0^K \frac{\alpha_i}{\beta_0} y_{n-i} .$$

The coefficient of y_n in the middle member above is $1 + 1/2 + 1/3 + \ldots + 1/K$, and so (using $\alpha_0 = -1$)

$$\beta_0 = 1 \Big/ \sum_{j=1}^K \frac{1}{j} .$$

The key to the success of these methods, as already stated, is their stiff stability. Before looking at the actual absolute stability regions for the methods, we can give a heuristic argument for selecting them, on the basis of a need for stiff stability. Consider a general implicit linear multistep method, with its associated polynomials ρ and σ, and roots $\xi_j(h\lambda)$ of $\rho + h\lambda\sigma$. When solving a stiff problem, with a value of h that is comparable to the larger time constants of the problem, we know that for some eigenvalue λ of the Jacobian, $h\lambda$ has a highly negative real part. We want our method to have an S_a that contains such points $h\lambda$. We can think of $h\lambda$ as lying near the "point" at $-\infty$, meaning that it is indefinitely far to the left in the $h\lambda$ plane. Thus we can approximate $\xi_j(h\lambda)$ by $\xi_j(-\infty)$, the limiting values of the roots as $h\lambda \to -\infty$. These limits actually exist, and are given by the roots of $\sigma(\xi)$. This is because when $|h\lambda|$ is large, the ρ in $\rho + h\lambda\sigma$ can be neglected, leaving only $h\lambda\sigma(\xi)$, which has the same roots as $\sigma(\xi)$. (This reasoning is only heuristic, but can be made rigorous.) The polynomial $\sigma(\xi)$ for Gear's method of order K is simply

$$\sigma(\xi) = \beta_0 \xi^K,$$

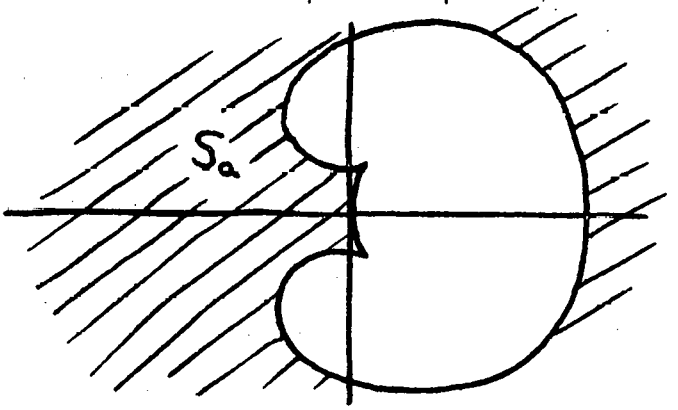
and all the roots of this are zero,

$$\xi_j(-\infty) = 0.$$

Roots which are zero are the best one can hope for, as this means that, in the limit, there is <u>no</u> propagation of any perturbations in the calculated values. The idealized point $-\infty$ is usually called simply "the point at infinity." The same limiting values $\xi_j = 0$ are obtained no matter how $|h\lambda|$

becomes arbitrarily large. Since the region S_a for the method is defined as the set of all $h\lambda$ for which all $|\xi_j(h\lambda)| \leq 1$, we see that the point at ∞ lies well within S_a. This statement can be taken to mean that S_a contains all points outside of some finite circle. Thus, whatever else S_a contains, it contains points that are arbitrarily distant from the origin in any direction. This is an indication (but not a proof) that it will satisfy the condition of stiff stability.

In actual fact the regions S_a for various K have been plotted quite accurately. They are shown on pp 215-216 in the text. In a typical case, the region consists of all the points <u>outside</u> a certain finite closed curve, as in the figure. In the left half-plane, which we are primarily concerned with, the closed curve has two lobes, appearing symmetrically about the horizontal axis. For K = 1, 2, ..., 6 these lobes come further and further to the left. For $K > 6$, they cross each other, and in doing so remove part of the real axis from S_a. This makes the methods no longer stiffly stable, and so those values of K are not generally used. For $1 \leq K \leq 6$, the methods are stiffly stable, as can be seen from the plots of S_a.



These methods are implemented in the (rather lengthy) program listed in the text, on pp 158-166. They are also implemented in a heavily modified

version of the program in the text, available in the report UCID-30001, Rev. 2.* In both programs, the principal features include the following:

- Gear's methods of orders K = 1, 2, ..., 6 (K = 1, 2, ..., 5 in the modified version);
- The implicit Adams methods as an option, of orders K = 1, 2, ..., 7 (K = 1, 2, ..., 12 in the modified version);
- The Nordsieck history for storage of past values;
- The chord method or functional iteration as options for solution of the implicit equation;
- Automatic startup;
- Automatic change of step size and order based on estimated local truncation error.

Another significant feature of these two programs is the way in which a change of step size h is accomplished. When the value is h at a given step, and the subsequent steps are to be done with a different value h', the program effectively interpolates, using the data with a spacing of h, to approximate the needed data with a spacing of h'. This process can actually be a source of instability, since the theoretical properties of the methods used are derived on the basis of a fixed value of h. This instability does not arise if, when running at order K, h is not changed for at least K consecutive steps.

---

*A. C. Hindmarsh, "GEAR: Ordinary Differential Equation System Solver," UCID-30001, Rev. 2, August 1972.

But changes in h may be forced to occur, for example by a failure to pass the error test, more frequently than that. To the extent that this happens, there is a potential for unstable behavior.

12. (d) The Liniger-Willoughby methods*

The problem of stiffness can arise in many practical problems. The sizes (numbers of equations in the system) can vary from many thousands all the way down to one. Of course, for the larger sizes, special techniques, such as sparse matrix methods, must be brought into use, in order to implement the appropriate kinds of ODE methods.

To illustrate the occurrence of stiff problems in practice, the following are three problems, which are rewritten in a neater and more abstract form than when originally posed, but which arose from realistic engineering situations.

(1) The percentage of delayed neutrons in a reactor is governed by a system such as the following:

$$\dot{x} = a(y-x)$$

$$\dot{y} = (b-d)x - (c-d)y + \alpha t(y+1)$$

$$0 < d < a \ll b < c, \quad \alpha \ll 1, \quad x_0 = y_0 = 0$$

E.g. $a = \frac{1}{2}$, $d = \frac{1}{10}$, $b = 10$, $c = 60$, $\alpha = \frac{1}{5}$

$$0 \le t \le t_f, \quad t_f \ge 400$$

At $t = 0$, eigenvalues $\lambda = -\frac{1}{2}, -60$.

---

*This lecture was given by Ralph A. Willoughby of I.B.M. in Yorktown Heights, New York. Most of the material can be found in W. Liniger and R. A. Willoughby, "Efficient Integration Methods for Stiff Systems of Ordinary Differential Equations," SIAM J. on Numerical Analysis, Vol. 7 (1970) pp 47-66.

Here the αt term represents the pulling out of a control rod from the reactor, which goes from subcritical to critical, and slightly supercritical. The complete problem involves controlling the parameter α so as to control the supercriticality to stay below acceptable levels. When this problem was attempted with the Runge-Kutta method, it was found that step sizes that seemed reasonable were always too large to give acceptable answers.

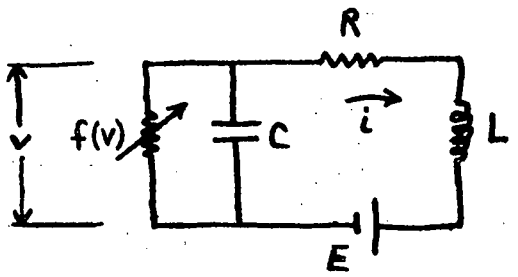
(2) A problem in enzyme kinetics can be put into the following form:

$$\dot{x} = -\alpha + uv$$
$$\dot{y} = \alpha - uw, \qquad x_0 = y_0 = 0$$
$$u = y - x + \alpha, \quad v = \beta(1-x) + x^2, \quad w = 1 + y^2$$
$$0 < \alpha << 1 << \beta$$
$$\dot{u} = 2\alpha - u(v+w)$$
$$t_f > 100$$
$$\text{At } t = 0, \text{ eigenvalues } \lambda = -.9 \cdot 10^{-3}, -11.$$

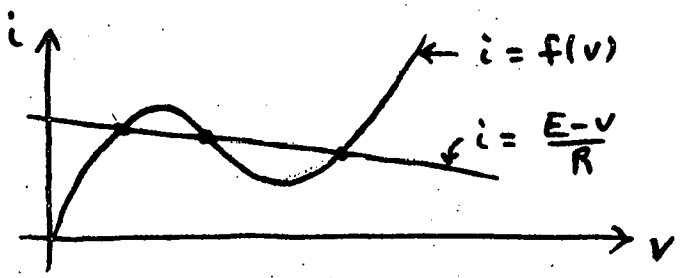
The ODE for u shows an exponential-like behavior common to most of these problems. However, here the relevant coefficient v + w is not a constant, but varies with the solution of the complete problem. In this case, the problem itself has an instability, in that a single inaccurate time step can lead into another solution region and so to drastically incorrect final answers.

(3) The tunnel diode, shown schematically here, is governed by an equation of the form

$$\frac{di}{dv} = \frac{C}{L}\,\frac{E-v-Ri}{i-f(v)}\,.$$



The equilibrium solutions of this equation correspond to stable states of the diode. There are typically three of these, of which two are stable and one is unstable. These are shown as the intersections of the curve $i = f(v)$ and the line $i = (E-v)/R$, the "load line." The unstable state is the intersection in the middle, where the curve has a negative slope. It has the properties of a saddle point. This problem exhibits what is called "bistability," in that starting values on one side of a certain curve (called the separatrix) lead to one equilibrium state, and starting values on the other side lead to the other equilibrium state.



When problems such as these arise, and it is found that they pose difficulties in their numerical solution, it is not a good idea just to work out clever techniques tailored to the particular problem to get answers for that problem. What is needed is the development of methods which can

effectively handle whole classes of problems, of which the above are only examples. Therefore, in what follows, we discuss general methods for the class of stiff problems.

Write the given ODE as*

$$\dot{x} = F(x),$$

where x, $\dot{x}$, and F are vectors, and $\dot{x} = dx/dt$. We omit any explicit dependence on time t in the right-hand side function F, because including it does not add any insight into the nature of stiffness or how to deal with it. Problems which have this form (i.e. have to explicit dependence on the independent variable) are called antonomous. We note that for this ODE we can write the second derivative,

$$\ddot{x} = J(x)\, F(x),$$

where J is the Jacobian of the problem,

$$J(x) = \frac{\partial F}{\partial x} = \left(\frac{\partial F_i}{\partial x_j}\right).$$

The methods of Liniger and Willoughby, in their simplest form, are base on the idea of using a weighted average of the forward and backward Euler methods. The formula for the methods to be considered initially is the following, for the step from t_n to $t_{n+1} = t_n + h$:

$$0 = x_{n+1} - h(1-\mu)\dot{x}_{n+1} - (x_n + h\mu\dot{x}_n).$$

Here μ is a constant which is constrained by $0 \leq \mu \leq \frac{1}{2}$.

---

*The notation used in this section differs slightly from that used elsewhere, but is self-consistent.

This formula is implicit for any value of the constant μ, because of the term $\dot{x}_{n+1} = F(x_{n+1})$. The price of solving such implicit equations seems to be unavoidable if stiff systems are to be solved efficiently. We treat this implicit equation as a general nonlinear algebraic problem, and we solve it by Newton's method. This means that the equation is linearized with respect to x, and changes Δx in x are made according to

$$(J - \alpha I)\Delta x = -\alpha G(x).$$

Here α is the scalar

$$\alpha = \frac{1}{h(1-\mu)}.$$

The function G represents the residual amount by which x fails to satisfy the given formula. If $x_{n+1}^{(k)}$ is the k^{th} iterated approximation to x_{n+1}, then G is given by

$$G(x_{n+1}^{(k)}) = x_n + \mu h \dot{x}_n + (1-\mu) h \dot{x}_{n+1}^{(k)} - x_{n+1}^{(k)},$$
$$\dot{x}_{n+1}^{(k)} \equiv F(x_{n+1}^{(k)}).$$

The classical way of solving implicit formulas such as this was simply to take Δx to be G(x), which means taking

$$x_{n+1}^{(k+1)} = x_n + \mu h \dot{x}_n + (1-\mu) h F(x_{n+1}^{(k)}).$$

The trouble is that this iteration fails to converge for stiff problems, because J is somehow too large. In the Newton iterations, we divide by $J - \alpha I$, and this tends to eliminate exactly those errors in x that caused the convergence failures before.

In solving general problems, we can always think of the simple problem $\dot{x} = -\lambda x$ as a model. Here the true solutuion satisfies

$$x(t+h) = e^{-q} x(t), \qquad q = \lambda h .$$

We take $\mathrm{Re}(\lambda) \geq 0$, so that the solution is decaying. (Here λ represents the negative of an eigenvalue in the general problem.) What the method actually gives for this problem is

$$x_{n+1} = R(q,\mu)\, x_n ,$$

$$R(q,\mu) = \frac{1-\mu q}{1+(1-\mu)q} .$$

The function R therefore represents a rational approximation to the function e^{-q}. It has the desirable property that $|R| < 1$ for $\mathrm{Re}(q) > 0$, which means that the computed solutions decay with n. However, except for $\mu = 0$, R does not have the property

$$|R| \to 0 \quad \text{as} \quad \mathrm{Re}(q) \to \infty,$$

which is also desirable because it makes the decay rate of the computed solutions behave like that of the true solution (namely very rapid) when Re(q) is large. The special case $\mu = 0$ does give this property, and this corresponds to the backward Euler method.

At this point the idea of exponential fitting comes in. The value of μ, which we are free to choose, can be chosen on the basis of the value of q, so as to make the value of R just what it should be. More specifically, if we knew a real value of $q = h\lambda$, or a good approximation to it, then we can clearly solve for a value of μ that makes

$$R(q,\mu) = e^{-q} .$$

When this is done, the computed solution will decay in a manner that agrees very well with that of the solution $e^{-\lambda t}$. We can regard the case $\mu = 0$ as that of fitting at $q = \infty$.

Another value of μ that has special significance is $\mu = \frac{1}{2}$. This gives the trapezoid rule, which is a second order method, and it corresponds to fitting R to e^{-q} at $q = 0$ to second order. All other values of μ give methods that behave like the Euler methods in the limit of small q, in that they are only of order 1. But the choice $\mu = \frac{1}{2}$ causes a problem for large q that is like the Gibb's phenomenon in Fourier analysis. We see that

$$R(q, \tfrac{1}{2}) \to -1 \quad \text{as} \quad q \to \infty,$$

and the value -1 means that perturbations in the solution tend to remain undamped and to oscillate in sign. These perturbations have to be filtered out in some way in order to get accurate answers.

In the above exponential fitting process, the formula allows for a fit for any given real value of q. But in practice, the eigenvalues $-\lambda$ can be complex. So we consider the possibility of fitting the formula to a given complex constant q. Unfortunately, this is not possible with these formulas for any real coefficient μ. A natural extension of the same basic idea which does make this possible is to use second derivative terms in the formula. This is not as difficult as it may seem, because $\ddot{x} = JF$, and J is already required for the Newton iteration. With $\ddot{x}$ terms, the Newton iteration will then involve a matrix which is quadratic in J rather than linear.

The expanded formula in its general form is

$$0 = \left[x_{n+1} - \frac{h}{2}(1+a)\dot{x}_{n+1} + \frac{h^2}{4}(b+a)\ddot{x}_{n+1}\right] - \left[x_n + \frac{h}{2}(1-a)\dot{x}_n + \frac{h^2}{4}(b-a)\ddot{x}_n\right],$$

where a and b are two free constants. This can be regarded as a weighted average of the two formulas gotten by writing Taylor series forward from t_n to t_{n+1} and backward from t_{n+1} to t_n, and neglecting terms that are $O(h^3)$. For the simple equation $\dot{x} = -\lambda x$, we find that

$$x_{n+1} = R(q, a, b)\, x_n \,, \qquad q = \lambda h \,,$$

$$R(q,a,b) = \frac{1 - \frac{1}{2}(1-a)q + \frac{1}{4}(b-a)q^2}{1 + \frac{1}{2}(1+a)q + \frac{1}{4}(b+a)q^2} \,.$$

One special case of interest is that with a = 0, b = 1/3. The resulting formula,

$$x_{n+1} = x_n + \frac{h}{2}(\dot{x}_n + \dot{x}_{n+1}) + \frac{h^2}{12}(\ddot{x}_n - \ddot{x}_{n+1}),$$

is an extension of the trapezoid rule to $\ddot{x}$ terms. It has an R of

$$R(q, 0, \tfrac{1}{3}) = \frac{1 - q/2 + q^2/12}{1 + q/2 + q^2/12} \,,$$

which is the second diagonal Padé approximation to e^{-q}. However, we see that as $q \to \infty$, we get a limiting value of 1 for R. This means that perturbations remain undamped and of constant sign. This is even worse than with the trapezoid rule, because these purturbations cannot be filtered out.

One solution to this difficulty is to take $b = a > 0$, so that R reduces to

$$R(q, a, a) = \frac{1 - \frac{1}{2}(1-a)q}{1 + \frac{1}{2}(1+a)q + \frac{1}{2}aq^2} .$$

Now $R \to 0$ as $q \to \infty$, and the damping properties are as desired.

For oscillatory problems we want to consider pure imaginary values of q. We find that the choice $a = 0$ makes the two coefficients of q^2 equal in the numerator and denominator of R, and so makes $|R| = 1$ for any pure imaginary q.

All of these special cases are examples of exponential fitting with the general formula, by appropriate choices of the free parameters. If b is fixed, or constrained by $b = a$, for example, the choice of a may be made to satisfy $R = e^{-q}$ for a given real value of q. If a and b are both left free, they can be chosen to make $R = e^{-q}$ for two distinct real values of q, or for two complex conjugate nonreal values. The reactor problem (1) given at the beginning of the section was solved by letting $b = \frac{1}{3}$ and choosing a to match the larger eigenvalue (-60), with a step size of 5. The corresponding value of q is 300.

There is a difficult task in the control of errors for these stiff problems. This is because there is a high degree of cancellation of terms in the evaluation of F. There does not seem to have been a decent study made of this difficulty.

The Newton iteration for the formula with $\ddot{x}$ terms present leads to, as stated earlier, a matrix involving J^2 as well as J. By factoring this quadratic in J, the linear system problem there can be rephrased as one of solving a system of the same form as before

$$(J - \alpha I)\Delta x = -\alpha G ,$$

but where α is complex, and the imaginary part of Δx is what is needed. This eliminates the need to form J^2.

It should be emphasized that all of the formulas discussed here are one-step formulas. This is an advantage in that errors in past values do not affect the behavior of the calculation on the current step, as is the case with parasitic roots in a multistep method. On the other hand, the one-step nature of the methods prevents them from attaining high orders of accuracy, which are often desirable.

The parameters a and b are not entirely free if certain stability and accuracy conditions are imposed. The condition of A-stability ($|R| < 1$ for $Re(q) > 0$), for example, imposes the conditions $a > 0$, $b > 0$, and for any such values the formula is in fact A-stable. Other constraints arise if the local truncation error is examined. For example, it is desirable to have $b \geq a$ and $a < 1$.

A variation on the basic idea presented here is to use more than one version of the formula on each step and use the resulting computed values to eliminate the errors in any of them. This is the same approach as taken in extrapolation methods, except that here h is not made smaller and smaller. We simply use each of a small collection of pairs (a,b), and take an

appropriate linear combination of the several computed values x_{n+1}. This approach was studied by Liniger and Odeh.

A further variation of the formula is one being explored by W. Enright, in which further $\dot{x}$ terms are added. Specifically the formula is

$$x_{n+1} = x_n + h(\beta_0 \dot{x}_{n+1} + \beta_1 \dot{x}_n + \cdots + \beta_K \dot{x}_{n+1-K}) + h^2 \gamma_0 \ddot{x}_{n+1}.$$

Notice that this is a K-step formula, so that higher orders are possible.

The free coefficients above can also be chosen on the basis of stability. In particular, it is relatively easy to attain the condition of stiff stability (see Section (b) above). This property is nearly as attratice as A-stability, even if there are rapid oscillations, since the absolute stability regions go sufficiently far out along the imaginary axis (e.g. to about $\pm$ 2i).

With a proper choice of the β_i and γ_0, this method can be made to behave like the Adams methods for small q, i.e. to have as high an order as needed. At the same time, it behaves like the backward Euler method at $q = \infty$, because only the forwardmost second derivative term, $\ddot{x}_{n+1}$, is present. This has the desirable effect of producing the correct damping for large values of Re(q).

Again the idea of exponential fitting can be brough in. Instead of using fixed values of the coefficients, they can be chosen so as to fit given (or estimated) values of e^{-q}.

We close this section with two comments about the way that basic methods are implemented. These apply to discrete ODE methods in general, not just the ones of this section.

First, when a given method is accompanied by an estimate of the local error, it is not always helpful to add this error estimate into the calculated result. This seems contradictory, because such an addition would appear to give more accurate answers. The trouble is that, especially with stiff problems, this can destroy the stability properties of the original method. We can, however, do this addition to get values used for output, as long as those values are not also used in the subsequent calculation with the method.

A second comment concerns the use of different methods on the different components $\dot{y}_i = F_i$ of the ODE system. This is sometimes done to the extent of allowing a method of a different order on each component. In a general purpose ODE solver, it is very difficult to do this in a way that gives meaningful answers as efficiently as is done with the same method for all components.

Summary Remarks

A wide variety of methods, for the numerical solution of the initial value problem for ODE systems, has been discussed here. In addition, something of the variety of ODE problems that occur has been indicated. In the face of this great variety, it would be ideal if one method were best for all problems. That, of course, is far from being the case. Lowering our ideals somewhat, we might at least hope that for any given problem, it is obvious which method is the best for solving it. Unfortunately, that is also not the case. But in spite of this, there are a few general statements that seem to hold true for the most part, in the matter of choosing the best method for a given problem. These comments appear in Chapter 12 of the text.

(1) Trivial problems. Here we consider problems that are small in size, simple (in terms of the complexity of f), and involve no special difficulties such as stiffness. For such problems, human time is a more important consideration than computer time. Therefore, one should use whatever method or program is readily available and easy to use. A large sophisticated package is not called for here.

(2) Smooth non-stiff problems. For a problem of this type, assuming that size or complexity removes it from the group of trivial problems, computational efficiency is a major consideration. In this regard, the Bulirsch-Stoer (rational extrapolation) and various linear multistep methods are considered best. Of the latter, the Adams-Moulton methods are probably

the most effective. If the size of the problem, N, is large, then the linear multistep methods of high order are to be discouraged because of the storage problem. If the expense of f is large, then it is best to use an implementation of either the Bulirsch-Stoer or a group of linear multistep methods which allows both the step size and the order to change dynamically under some automatic control algorithm.

(3) Problems with discontinuities. For these problems, one should avoid multistep methods, unless the problem is restarted at each discontinuity. If the discontinuities are both frequent and sizable, it is better to use a one-step method instead, and force the discontinuities to fall on mesh points if the order of the method is higher than that of the discontinuity. The Bulirsch-Stoer and explicit Runge-Kutta methods are good choices here.

(4) Stiff problems. The methods of Gear are a good choice in general here. However, if there are frequent and sizable discontinuities, other methods should be considered, such as the Liniger-Willoughby or implicit Runge-Kutta methods, although they may be difficult to use in a general setting.

Some valuable information on the matching of the best methods to given problems is provided in the results of comparative testing that has been done with ODE solvers. One such series of tests was done by Hull et al.* They tested a group of programs, in which the Runge-Kutta, Adams, and

---

*T.E. Hull, W.H. Enright, B.M. Fellen, and A.E. Sedgwick, "Comparing Numerical Methods for Ordinary Differential Equations," SIAM J. on Numerical Analysis, Vol. 9, No. 4 (December 1972), pp 603-637.

Bulirsch-Stoer methods were used, on a group of initial value problems. The tests ignored difficulties caused by discontinuities, stiffness or startup (for multistep methods). The major conclusions of this project were summarized as follows:

"One conclusion is that the best general-purpose methods for nonstiff systems without discontinuities, and without considering any special starting difficulties, must be variable-order methods. A second conclusion is that, if function evaluations are not very costly, the best method of those tested is one due to Bulirsch and Stoer; however, when function evaluations are relatively expensive, variable-order methods based on Adams formulas are best. The overhead costs are lower for the method of Bulirsch and Stoer, but the Adams methods require considerably fewer function evaluations. Krogh's implementation of a variable-order Adams method is the best of those tested, but one due to Gear is also very good. Our third conclusion is that Runge-Kutta methods are not competitive in general, although fourth or fifth order methods of this type are best for restricted classes of problems in which function evaluations are not very expensive and accuracy requirements are not very stringent. These conclusions appear to hold uniformly over a wide variety of problems."

A consideration that these statements tend to neglect is the consideration of what programs for ODE's are <u>available</u>. Ideally, a user should have available a large battery of good programs, representing all method classes. Then once a method is chosen, it is not necessary to program it, but just to

use the appropriate code. Few if any installations are in that good a position, although at LLL we do have codes representing most method classes. An encouraging statement on this point that appears in the article quoted above is the following: "If a program library was to contain only one program for solving ordinary differential equations, we would strongly recommend Gear's. It is a good general-purpose method, in the sense used in this report, and it can also handle stiff systems very effectively." The program referred to is the one listed on pp 158-166 of the text, and a version of it is available at LLL.

Bibliography

Text:

1. C. William Gear, Numerical Initial Value Problems in Ordinary Differential equations, Prentice-Hall, 1971. Has an extensive but uncategorized bibliography.

General:

2. J.W. Daniel and R.E. Moore, Computation and Theory in Ordinary Differential Equations, Freeman, 1970.
3. E. Isaacson and H.B. Keller, Analysis of Numerical Methods, Wiley, 1966, Chap. 8.
4. L. Lapidus and J.H. Seinfeld, Numerical Solution of Ordinary Differential Equations, Academic Press, 1971. Has a reference list for each chapter.
5. A. Ralston, A First Course in Numerical Analysis, McGraw-Hill, 1965, Chap. 5.
6. J.R. Rice (ed.), Mathematical Software, Academic Press, 1971, Chapters 5.14 (by C.W. Gear), 5.22 (Taylor series methods, by Barton, Willers, and Zahar) and 9 (extrapolation method, with code, by P.A. Fox).
7. J. Todd, Survey of Numerical Analysis, McGraw-Hill, 1962, Chap. 9.
8. J.B. Scarborough, Numerical Mathematical Analysis, Johns Hopkins, 1966, Chap. XIII.
9. J. A. Zonneveld, Automatic Numerical Integration, Math. Centrum, 1970. Has ALGOL codes.
10. J. Walsh (ed.), Numerical Analysis: An Introduction, Thompson, 1966, Chapters 4 (Survey article by J.C.P. Miller), and 11 (Part II) (application by H.H. Robertson).

Runge-Kutta Methods:

11. Z. Ceschino and J. Kuntzman, Numerical Solution of Initial Value Problems, Prentice-Hall, 1966.

12. E. Fehlberg, "Klassische Runge-Kutta Formeln vierter and niedrigerer Ordnung ...", Computing, Vol. 6 (1970) pp. 61-71.

13. E. Fehlberg, "Klassische Runge-Kutta Formeln fünfter und siebenter Ordnung ...", Computing, Vol. 4 (1969) pp 93-106.

14. L.F. Shampine and H.A. Watts, "Efficient Runge-Kutta Codes," SC-RR-615, Sandia-Albuquerque, 1970.

Linear Multistep Methods:

15. P. Henrici, Discrete Variable Methods in Ordinary Differential Equations, Wiley, 1962.

16. P. Henrici, Error Propagation for Difference Methods, Wiley, 1963.

17. F.B. Hildebrand, Finite Difference Equations and Simulation, Prentice-Hall, 1968.

18. A.C. Hindmarsh, "Linear Multistep Methods for O.D.E.'s: Method Formulations, Stability, and the Methods of Nordsieck and Gear," UCRL-51186, Rev. 1, 1972.

Extrapolation Methods:

19. W.B. Gragg, "On Extrapolation Algorithms for Ordinary Initial Value Problems," SIAM J. on Numerical Analysis, 2 (1965), pp 304-403.

20. R. Bulirsch and J. Stoer, "Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods," Numerische Mathematik, 8 (1966), pp 1-13.

Non-Discrete Methods:

21. T.N.E. Greville (ed.), Theory and Applications of Spline Functions, Academic Press, 1969, Chapter by F.R. Loscalzo on ODE's.

22. H. Knapp and G. Wanner, "LIESE, a Program for Ordinary Differential Equations Using Lie Series," Mathematical Research Center Report 881, Madison, 1968.

Boundary Value Problems:

23. P.B. Bailey, L.F. Shampine, and P.E. Waltman, Nonlinear Two Point Boundary Value Problems, Academic Press, 1968.

24. H.B. Keller, Numerical Methods for Two-Point Boundary Value Problems, Blaisdell, 1968.

25. S.M. Roberts and J.S. Shipman, Two-Point Boundary Value Problems: Shooting Methods, Elsevier, 1972.

Appendix I: Lecture Timetable

The following table gives a rough correlation between the chapters of these notes, the course lectures, and the sections in the text. Some lectures covered two or more chapters, and some chapters required two or more lectures. In addition, parts of some lectures were devoted to discussion of homework problems.

Chapters in Notes	Lecture Numbers	Chapters/Sections in Text
1,2,3	1	1, 3.2, 4.1, 4.2
3	2,3,4	
4	4,5	
5	5,6	
6	7,8	4.3-4.7
7	9,10,11,12,13	5.1-5.3
8	14,15,16,17,18,19 (Lecture 18 by R. L. Pexton)	2.3-2.6, 4.6.2, 5.4
9	20,21,22,23,24,25,26,27	7,8,9
10	28 (by J. S. Chang)	6
11	29	11.1
12	29,30,31 (Lecture 31 by R. A. Willoughby)	11.1
Summary	32	12

Appendix II: Errata in Text

The following is an ordered list of errata in the text, Numerical Initial Value Problems in Ordinary Differential Equations, by C.W. Gear. It is probably incomplete. Line numbers given are as counted from the top of the page, except that a negative line number, $-\ell$, refers to the ℓ^{th} line from the bottom of the page.

Chapter	Page	Line	Corrected Version
1	18	10	$e_N \tilde{=} \frac{-h}{1000}$
1	18	12	$(-1)^N e_N \tilde{=}$
1	20	4	range $[\frac{-\epsilon}{2}, \frac{\epsilon}{2}]$
1	20	5	$[-(N/2)\epsilon, (N/2)\epsilon]$
1	23	-1	numbers to 10^{-6}.
2	28	3	$-hf(y_n, t_n)$

Chapter	Page	Line	Corrected Version
2	34	(2.18)	$-\frac{h^4}{24}\Big[f^i_{jkl}f^jf^kf^l(1+2\alpha_1\alpha_2-\frac{4}{3}(\alpha_1+\alpha_2))$ $+f^i_{jk}f^j_lf^kf^l\ (3-4\alpha_2)$ $+f^i_jf^j_{kl}f^kf^l\ (1-2\alpha_1)$ $+\ \ldots$
2	38	-4	is linear in β_{12} and β_{22}.
2	39	3	$hf(y_n+\frac{2}{3}k_1)$
4	55	(4.8)	$\|\|\underset{\sim}{f}(\underset{\sim}{y},t) - \underset{\sim}{f}(\underset{\sim}{y}^*,t)\|\|$ [boldface $\underset{\sim}{f}$, $\underset{\sim}{y}$]
4	55	-11, -15	$f^i(\underset{\sim}{y},t) - f^i(\underset{\sim}{y}^*,t)$
4	56	5	$\underset{\sim}{y}_n \to \underset{\sim}{y}(t)$
4	56	7	$\underset{\sim}{y}' = \underset{\sim}{f}(\underset{\sim}{y},t)$
4	56	-8	condition in y
4	57	12	condition in $\underset{\sim}{y}$

Chapter	Page	Line	Corrected Version
4	58	21	$\underset{\sim}{f}(\underset{\sim}{y}(t),t)$
4	59	-10	$\frac{h^2}{6}\frac{df_n}{dt}$, f_n +
4	60	16	$\underset{\sim}{e}_{n+1} = \ldots$
4	61	-9	$\max_{0 \le t \le 1} \left\| \frac{y^{(4)}(t)}{6} \right\|$
4	61	-11	ξ_n' [instead of second ξ_n]
4	62	4	$h^{r+1}\,\underset{\sim}{\phi}(\underset{\sim}{y},t)$
4	62	23	(Theorem 4.4)
4	62	-5	$\underset{\sim}{\delta}_n$ converges to $\underset{\sim}{\delta}(t_n)$ [boldface]
4	65	16	$=(r+1)\tau^r \underset{\sim}{T}(\ )$
4	66	10	region of $\underset{\sim}{y}$-space
4	67	-9	$f_j^i(\underset{\sim}{y}+\frac{h}{2}\underset{\sim}{l}(\underset{\sim}{y}))f^j(\underset{\sim}{y})$
4	69	17	show that (4.30) implies

Chapter	Page	Line	Corrected Version
4	69	21	$+kh^2\theta(t_n)$
4	69	-10 to -7	$\theta(t_i)$ [instead of $\theta(t_n)$]
4	69	-11	$+\theta(t_i)hL$ [under product sign]
5	76	9	$\sum_{n=0}^{N-1} \int$
5	76	16	in the interval
5	76	-7	$\| \underset{\sim}{e}_N \| \int$
5	77	16	$\frac{1}{\|G(t_n,b)\| \, \|\phi(y,t_n)\|}$
5	77	-5	hold for $\epsilon \neq 0$,
5	84	22	15.0D0)
5	84	28	(16.0D0 * Y3(I) - Y1(I))/15.0D0

Chapter	Page	Line	Corrected Version
5	85	-8	$f_{ijk}f^{i}f^{j}f^{k}$, $f_{i}f^{i}_{jk}f^{j}f^{k}$, and $f_{ij}f^{i}_{k}f^{j}f^{k}$.
5	86	-9	EPS*H* |YMAX(I)| instead of EPS* |YMAX(I)| . Plot ...
6	93	-2	$\nu = m - \mu =$
6	93,99		[Interchange Tables 6.5, 6.7.] [Fourth number in column 0 should be 917384 in both tables.] [Text on middle of p. 99 is now incorrect. Rational extrapolation is better than polynomial extrapolation for this example also.
6	95	(6.12)	$P_{n+1}(h) - yQ_{n+1}(h) = \ldots$ ... $-y[\ \ldots]$
7	106	2	$\binom{s}{0} = 1.$
7	106	3	(7.5)
7	111	14	$\xi \in (t_{n-k}, t_n)$

Chapter	Page	Line	Corrected Version
7	113	-16	There are two ...
8	117	-1	the Taylor series for $L_h(y)$ vanish.
8	120	-4	$\ldots - \frac{1}{12} hy'_{n-2} = 0$
8	123	Table 8.3	$y(0) = 1$ [in title]
8	123	"	[Change sign of entries under Error]
8	124	7	exponentially; ... linearly;
8	125	19	$\left\| \xi_i(0) \right\| = 1$
8	118	8	$0(h^{r+2})$
9	137	4	$+\frac{3}{8}$ [hf...

Chapter	Page	Line	Corrected Version
9	137	-13	$\underset{\sim}{d}_n = \underset{\sim}{\tilde{y}}_n - \underset{\sim}{y}(t_n)$
9	138	18	on the eigenvalues
9	139	5	$\ldots\ b_k\rceil$
9	139	-5	eigenvalues
9	140	7	less in magnitude than
9	140	-1	$\ldots\epsilon_i\ldots \quad \vdots \quad \ldots b_i\ldots\rceil$
9	141	15	$\xi^k \rho^*(\xi) = 0.$
9	141	17	eigenvalues
9	141	24	$hy_i' = hf(y_i)$,

Chapter	Page	Line	Corrected Version
9	141	27	for $i<k$, $hy_i' \neq$
9	143	11	it is useful to state:
9	145	2	$\tilde{hy}'(t-\tilde{h})=$
9	147	5	were seen to be
9	147	(9.11)	and $\underset{\sim}{\ell} = \underset{\sim}{Tc} =$
9	147	-13	is not as large as it might
9	148	8	$\begin{bmatrix} 1 & \gamma_0 & \gamma_1 & \cdots \gamma_{k-2} \end{bmatrix}$
9	148	-8	Nordsieck
9	149	10	premultiplying by A

Chapter	Page	Line	Corrected Version
9	149	5	given by the B and $\underset{\sim}{c}$ preceding (9.11),
9	149	16	$O(h^K)$.
9	150	-10	Eq. (9.2)
9	150	-5	of degree three
9	152	14	$\prod_{i=0}^{M-1}$ [not $\sum$]
9	162	following statement no. 217	A(3) = -1.225
9	168	3	6. Equation (9.12)
10	192	-7,-12	$\frac{\partial F^i}{\partial \underset{\sim}{a}^j}$ [incorrect denominator]
10	192	-2	[(10.43) should be on <u>last</u> line.]

Chapter	Page	Line	Corrected Version
10	193	(10.46)	$e^{i}_{\mathbf{N},}(m)$
10	204	-17	$O(h^{r+2})$.
10	204	-13	$f_y(\xi_{n-i})$
10	204	-12	$\beta_i y^{(r+1)}(t_n) = \beta_i y^{(r+1)}(t_{n-i}) +$
10	204	-7	$g(t)\delta(t) +$
10	204	-6	$g(t) = f_y(y(t))$.
11	227	13	$\frac{\partial E}{\partial \underset{\sim}{y}} \ell_0 + \frac{\partial E}{\partial \underset{\sim}{y}'} \frac{\ell_1}{h}$ [ℓ is not boldface]
11	227	17	Tewarson

:aas

DISTRIBUTION

LLL Internal

T. I. D. File (5 copies)	L-009	J. G. Huebel	L-156
J. J. Brandt	L-307	S. L. Jenkins	L-724
J. Breazeal	L-307	E. D. Jones	L-152
J. S. Chang	L-071	J. I. Karush	L-310
R. C. Chin	L-051	N. K. Madsen	L-310
L. M. Cook	L-024	D. E. Maiden	L-090
J. L. Cramer	L-402	A. D. Pasternak	L-437
C. S. Denton	L-123	R. L. Pexton	L-310
R. P. Dickinson	L-310	R. Quong	L-437
C. H. Finan	L-388	H. H. Reed	L-396
F. N. Fritsch	L-310	P. A. Renard	L-216
D. Fuss	L-318	N. J. Roberts	L-435
R. J. Gelinas	L-071	J. M. Sayer	L-388
P. M. Gresho	L-142	D. R. Slaughter	L-520
G. L. Hage	L-307	T. W. Stullich	L-142
W. J. Hannon	L-044	T. Suyehiro	L-024
R. J. Harrach	L-071	P. F. Thompson	L-071
A. C. Hindmarsh (20 copies)	L-310	J. B. Trenholme	L-216
B. C. Howard	L-318	P. P. Weidhaas	L-024
R. H. Howell	L-520	A. M. Winslow	L-071

External

J. Alcone
Division 5722
Sandia Laboratories
Sandia Base
Albuquerque, NM 87115

G. D. Byrne
Applied Mathematics Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

R. E. Huddleston
Division 8441
Sandia Laboratories
PO Box 969
Livermore, CA 94550

L. J. Leonard
542 Oakridge Drive
Rochester, NY 14617

W. Black
Mathematics and Physics Department
Savannah State College
Savannah, GA 31404

R. F. Sincovec
Computer Science Department
Kansas State University
Manhattan, KS 66506

J. F. Traub
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

T. Donlon
458 College Avenue, Apt. 203
State College, PA 16801

L. W. Mays
2604 S. Anderson Street
Urbana, IL 61801