RUNGE–KUTTA METHODS IN MODERN COMPUTATION, PART I: FUNDAMENTAL CONCEPTS

J. C. Butcher **Department Editor: William J. Thompson**

Part I of this two-part article discusses fundamental concepts such as the order of Runge–Kutta methods, arborescences, the use of tableaux, stiffness, and stability. The explicit Runge–Kutta method is illustrated by means of the forced-damped-oscillator problem. Part II, to appear in the next issue, will consider implicit Runge–Kutta methods, error estimation, implementation issues, and the solution of related problems: partial-differential and differential–algebraic equations.

Virtually every branch of physics is concerned with the rate at which something changes. Hence, the study of differential equations and the behavior of their solutions have always been of central interest to physical scientists. The rich mathematical field of special functions has grown from the need to solve particular differential equations arising from physical problems. However, most differential equations arising in scientific modeling do not have closed-form solutions. For this reason, numerical methods for differential equations have a central role in computational physics. Of particular interest is the special class of Runge–Kutta methods. These have enjoyed a popularity for physical computations because of their ease of use, their natural and intuitive structure, and the excellence of their stability and other computational properties.

In this survey of Runge–Kutta methods we will review their traditional role in practical computation and also discuss some recent developments. Throughout, we will concentrate on initial-value problems of the form

$$\frac{dy^i}{dx}=f^i(y^1,y^2,\dots,y^N), \quad i=1,2,\dots,N, \tag{1}$$

with initial conditions

$$y^i(x_0)=y_0^i, \quad i=1,2,\dots,N. \tag{2}$$

It will usually be convenient to write Eqs. (1) and (2) in vector form as

$$\frac{d\mathbf{y}}{dx}=\mathbf{f}(\mathbf{y}) \tag{3}$$

and

$$\mathbf{y}(x_0)=\mathbf{y}_0,$$

where

$$\mathbf{y}=\begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}, \quad \mathbf{f}=\begin{bmatrix} f^1 \\ f^2 \\ \vdots \\ f^N \end{bmatrix}, \quad \mathbf{y}_0=\begin{bmatrix} y_0^1 \\ y_0^2 \\ \vdots \\ y_0^N \end{bmatrix}.$$

Although systems of this type are necessarily of first order and autonomous in the sense that the time variable x does not occur explicitly as an argument of any of $f^1,f^2,\dots,f^N$, they can be adapted to cover a wider class of problems such as equations of second or higher order and problems in which x occurs on the right-hand sides.

This will be illustrated by the equation of forced damped oscillations

$$\frac{d^2z}{dx^2}+a\frac{dz}{dx}+bz=c\sin(wx), \tag{4}$$

with initial amplitude $z(x_0)=p$ and slope $z'(x_0)=q$, where $z'(x)$ denotes dz/dx. To write down an equivalent system of the form of Eq. (1), set $N=3$ and identify $y^1(x)$, $y^2(x)$, and $y^3(x)$ with $z(x)$, $z'(x)$, and x, respectively. The initial-value problem is then replaced by the system

$$\frac{dy^1}{dx}=y^2,$$

$$\frac{dy^2}{dx}=c\sin(wy^3)-ay^2-by^1, \tag{5}$$

$$\frac{dy^3}{dx}=1,$$

with initial values $y^1(x_0)=p$, $y^2(x_0)=q$, $y^3(x_0)=x_0$.

John Butcher is professor of mathematics at The University of Auckland in Auckland, New Zealand. He has made numerous contributions to the theory of Runge–Kutta methods and other aspects of the numerical solution of ordinary differential equations, and is the author of a definitive monograph (Ref. 1) on Runge–Kutta and general linear and methods. His current interest include the extension of singly implicit Runge–Kutta methods to the solution of differential–algebraic equations and the solution of ordinary differential equations in a parallel environment. The work in the present paper was supported by the New Zealand Foundation for Research, Science, and Technology.

Table I. A sample Runge–Kutta calculation.

n	y_{n-1}	$\mathbf{f}(\mathbf{Y}_1)$	$\mathbf{Y}_2$	$\mathbf{f}(\mathbf{Y}_2)$	$\mathbf{Y}_3$	$\mathbf{f}(\mathbf{Y}_3)$	$\mathbf{Y}_4$	$\mathbf{f}(\mathbf{Y}_4)$	$\mathbf{y}_n$
1	0.30000 4.00000 0.00000	4.00000 −19.10000 1.00000	0.70000 2.09000 0.10000	2.09000 2.13088 1.00000	0.50900 4.21309 0.10000	4.21308 4.95171 1.00000	1.14262 4.99034 0.20000	4.99034 −2.98505 1.00000	1.01988 3.73600 0.20000
2	1.01988 3.73600 0.20000	3.73600 4.06477 1.00000	1.39348 4.14248 0.30000	4.14248 −16.68342 1.00000	1.43413 2.06766 0.30000	2.06766 −14.03774 1.00000	1.43342 0.92846 0.40000	0.92846 −38.14391 1.00000	1.58938 0.55196 0.40000

Most numerical methods for Eq. (1) extend the solution forward in time using a step-by-step process. Suppose that, in additional to a given initial value $\mathbf{y}_0$, further vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n-1}$ have already been computed as approximations to $\mathbf{y}(x_0+h)$, $\mathbf{y}(x_0+2h), \ldots, \mathbf{y}[x_0+(n-1)h]$; then the aim of such a method would be to compute a further approximation $\mathbf{y}_n$ to $\mathbf{y}(x_0+nh)$. Note that h, the so-called "stepsize," measures how much the time variable increases per step.

Linear multistep and Runge–Kutta methods

Traditional numerical methods for performing this calculation come in two main classes, known as linear multistep methods and Runge–Kutta methods. In linear multistep methods, the vector $\mathbf{y}_n$ is approximated as a linear combination of a number of previously found step values $\mathbf{y}_{n-1}, \mathbf{y}_{n-2}, \ldots, \mathbf{y}_{n-k}$, together with scaled derivatives at the same points, $h\mathbf{f}(\mathbf{y}_{n-1}), h\mathbf{f}(\mathbf{y}_{n-2}), \ldots, h\mathbf{f}(\mathbf{y}_{n-k})$. In Runge–Kutta methods, on the other hand, $\mathbf{y}_n$ is found using a formula involving $\mathbf{y}_{n-1}$ but none of the earlier $\mathbf{y}_{n-2}, \mathbf{y}_{n-3}, \ldots$. The formula in question may be complicated and involve many evaluations of the $\mathbf{f}$ function. These evaluations are known as "stages," and, because evaluating $\mathbf{f}$ is typically an expensive computation, the number of stages is a measure of the complexity of a Runge–Kutta method.

In the best known example of a Runge–Kutta method, there are four stages. Denote by $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$, $\mathbf{Y}_4$ the four points at which $\mathbf{f}$ is evaluated. For this method, these quantities are defined as

$$\begin{aligned} \mathbf{Y}_1 &= \mathbf{y}_{n-1}, \\ \mathbf{Y}_2 &= \mathbf{y}_{n-1} + \frac{h}{2}\mathbf{f}(\mathbf{Y}_1), \\ \mathbf{Y}_3 &= \mathbf{y}_{n-1} + \frac{h}{2}\mathbf{f}(\mathbf{Y}_2), \\ \mathbf{Y}_4 &= \mathbf{y}_{n-1} + h\mathbf{f}(\mathbf{Y}_3), \end{aligned} \tag{6}$$

and $\mathbf{y}_n$ is given by

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \frac{h}{6}[\mathbf{f}(\mathbf{Y}_1) + 2\mathbf{f}(\mathbf{Y}_2) + 2\mathbf{f}(\mathbf{Y}_3) + \mathbf{f}(\mathbf{Y}_4)]. \tag{7}$$

To see how this method works in practice, consider the example of the forced damped oscillator Eq. (4), with $a=2$, $b=37$, $c=50$, $w=7$, and initial values $x_0=0$, $p=0.3$, $q=4$. We will carry out two steps with $h=0.2$, as shown in Table I.

The exact answer at $x=0.4$ is given by the column vector $[1.59481,\ 0.56374,\ 0.40000]^T$, where ^{T} denotes transpose. Not surprisingly, the error in the third component is zero, but is 0.3% and 2% approximately in the first and second components. To understand how the error depends on the number of steps taken to reach a given output point, see Fig. 1. Here the error in each of $y^1(0.4)$ and $y^2(0.4)$ is plotted for $h=0.4/n$, with $n=1,2,3,\ldots,100$ on a log–log scale. At least for large values of n, it seems clear that the error is close to being proportional to n^{-4}.

Order of Runge–Kutta methods

If a different Runge–Kutta method had been used, we might have found that the error was proportional to n^{-p} for

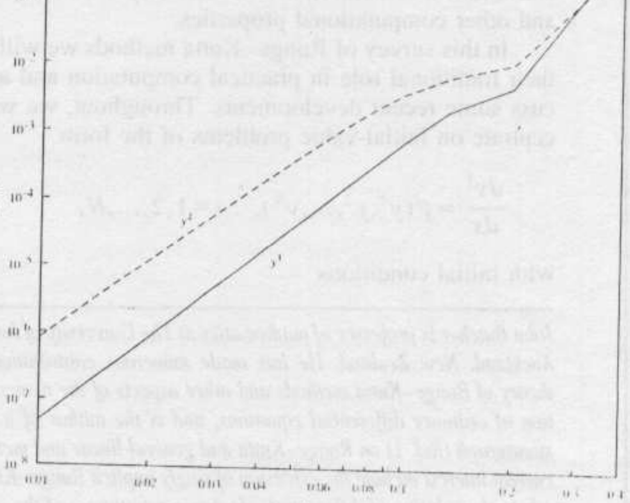

Figure 1. Error dependence on h for the Runge–Kutta method (6), (7) applied to the problem (4).

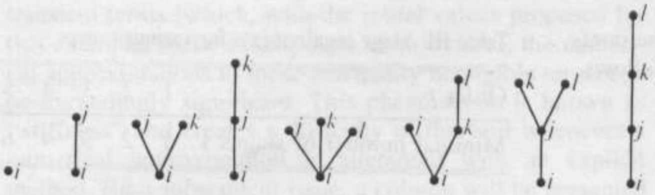

Figure 2. Trees representing elementary differentials of orders 1 through 4.

some different integer p. This number, known as the "order," is a characteristic of the particular Runge–Kutta method chosen.

Because $y'^i = f^i$, we find by differentiating that $y''^i = f^i_j y'^j = f^i_j f^j$ where we have written $f^i_j = \partial f^i / \partial y^j$, used the summation convention $f^i_j f^j = \Sigma^n_{j=1} f^i_j f^j$, and substituted $y'^j = f^j$. Following on from this formula for y'', we also find

$$y'''^i = f^i_{jk} f^j f^k + f^i_j f^j_k f^k$$
$$y^{(4)i} = f^i_{jkl} f^j f^k f^l + 3 f^i_{jk} f^j f^k_l f^l + f^i_j f^j_{kl} f^k f^l + f^i_j f^j_k f^k_l f^l. \quad (8)$$

Formulas for higher derivatives can be found in a similar manner, even though the details become exceedingly complicated as the order of differentiation increases. Evaluating the sequence of derivatives at $x = x_0$ leads in turn to the Taylor series for the (exact) solution at $x = x_0 + h$. To assess the order of a Runge–Kutta method, a similar series expansion is needed, but for the result at $x = x_0 + h$ as computed in a single step of the method.

If these two series agree up to terms in h^p, then the method has order p. This will mean that for a smooth problem, the error in a single step will be $O(h^{p+1})$. The accumulated error at a fixed value of x also has a behavior related to p. This is because the number of steps required to move from x_0 to x is proportional to h^{-1}. Because an error $O(h^{p+1})$ is generated in each of these steps, the total error will be $O(h^p)$. We will refer to the various terms occurring in y'^i, y''^i, y'''^i, $y^{(4)i}$, etc. (omitting the factor 3 that occurs in $y^{(4)i}$, and any similar factors that might occur for higher derivatives) as "elementary differentials."

Runge–Kutta trees

Before we explain the conditions for the agreement of the formal power series for the exact and computed result, we note an interesting fact about the structure of the elementary differentials. This observation is that they resemble family trees, where only descent from a single parent is recorded. For example, a factor f^i_j can be read as "j is the only offspring of i," whereas a factor f^j reads "j has no descendants." Similarly f^i_{jk} denotes i has two offspring named j and k. Using this idea, the eight terms occurring up to $y^{(4)i}$ are shown in diagrammatic form in Fig. 2, where we have attached letters $i, j, \ldots$ to the vertices to clarify the structure.

Table II. Some functions defined from Runge–Kutta trees.

$r(t)$	t	$\mathbf{F}(t,\mathbf{y}_0)$	$\sigma(t)$	$\gamma(t)$	$\Phi(t)$
1	·	f^i	1	1	$\sum b_i$
2	|	$f^i_j f^j$	1	2	$\sum b_i c_i$
3	V	$f^i_{jk} f^j f^k$	2	3	$\sum b_i c_i^2$
	|	$f^i_j f^j_k f^k$	1	6	$\sum b_i a_{ij} c_j$
4	Ψ	$f^i_{jkl} f^j f^k f^l$	6	4	$\sum b_i c_i^3$
	V	$f^i_{jk} f^j f^k_l f^l$	1	8	$\sum b_i c_i a_{ij} c_j$
	Y	$f^i_j f^j_{kl} f^k f^l$	2	12	$\sum b_i a_{ij} c_j^2$
	|	$f^i_j f^j_k f^k_l f^l$	1	24	$\sum b_i a_{ij} a_{jk} c_k$

Although these diagrams are technically known as "rooted trees" or "arborescences," we will use the simple word "trees." The lowest vertex in the trees as they are drawn here (in the family tree analogy, the oldest member of the family) is known as the "root."

Just as we can write down a tree corresponding to each elementary differential, we can also write down an elementary differential from each tree. It also happens that the series for the result computed by a Runge–Kutta method can be found in a similar way. If $\tilde{\mathbf{y}}$ denotes the computed result, the two series are

$$\mathbf{y}(x_0+h) = \mathbf{y}_0 + \sum_{t \in T} \frac{h^{r(t)}}{\sigma(t)\gamma(t)} \mathbf{F}(t,\mathbf{y}_0), \quad (9)$$

$$\tilde{\mathbf{y}}(x_0+h) = \mathbf{y}_0 + \sum_{t \in T} \frac{h^{r(t)}}{\sigma(t)} \Phi(t) \mathbf{F}(t,\mathbf{y}_0). \quad (10)$$

Various notations used here require some explanation. T is the set of all trees, t is a typical tree, $r(t)$ denotes how many vertices it has (the "order" of t), $\mathbf{F}(t,\mathbf{y}_0)$ denotes the elementary differential corresponding to t and evaluated at $\mathbf{y} = \mathbf{y}_0$, $\sigma(t)$ denotes the "symmetry" of t, and $\gamma(t)$ will be known as the "density" of t. Finally, the factor $\Phi(t)$ denotes a certain polynomial in the coefficients of the given Runge–Kutta method. Its occurrence in Eq. (10) indicates the manner in which the coefficients in a method influence its ability to approximate the exact solution (9) to a required order of accuracy.

From Table II, where these functions are displayed for the first eight trees, the meaning of the symmetry of t will be obvious. To calculate the density of t, use the family-tree analog again. With each vertex, associate an integer indicating the number of vertices in a subtree representing this person and his or her descendants alone. The product of these integers for all the $r(t)$ vertices in t is the density.

We now come to the function Φ. This is found in terms of coefficients a_{ij} indicating the coefficient of $h\mathbf{f}(\mathbf{Y}_j)$ in the expression for $\mathbf{Y}_i$ and b_j indicating the coefficient of $h\mathbf{f}(\mathbf{Y}_j)$

in the expression for $\mathbf{y}_n$. It is customary to write the coefficients of a Runge–Kutta method in a tableau as follows

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}, \tag{11}$$

where the c vector indicates the totals in the rows of A. For the classical method introduced in the previous section, the entries in the tableau (11) are

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}. \tag{12}$$

To calculate $\Phi(t)$, attach labels $i,j,k,\ldots$ to the vertices of t, except the vertices above the root (the so-called "leaves") from which no further vertices branch upwards. Having labeled the tree in this way, write down a factor b_i, where i is the label attached to the root of t, and a factor a_{jk} for all pairs of vertices j and k such that k branches upwards from j. Finally, insert a factor c_j for each leaf joined to a vertex j. Having written down all these factors, multiply them together and sum for each index $i,j,\ldots$ from 1 to s, the number of stages.

By comparing the series for $\mathbf{y}$ and $\tilde{\mathbf{y}}$, given by Eqs. (9) and (10), we can write down the conditions for agreement up to terms in h^p. This is

$$\Phi(t) = \frac{1}{\gamma(t)} \tag{13}$$

and is to hold for all trees up to the required order.

It is a simple matter to check that these conditions are satisfied up to order four for the classical method (12). This method has the sensible property that the matrix A is zero on and above the diagonal. If this were not the case, then the stages could not be evaluated in numerical order in a simple way. Methods like Eq. (12) are known as "explicit" Runge–Kutta methods, in contrast to those in which A has a more general structure which are said to be "implicit." We will discuss applications of implicit methods in the next section. In the meantime, we consider the question as to what order can be achieved with a given number of stages. For implicit methods this is easy: it is always possible to obtain an order $p = 2s$ when there are s stages. However, for the classical type of explicit method, the question is much more complicated.[1]

Table III. Stage requirements for various orders.

Order p	1	2	3	4	5	6	7	8
Minimal number of stages s	1	2	3	4	6	7	9	11

For $p = 1$, 2, 3, and 4, it is possible to obtain this order with $s = p$ stages, and this is the best that can be achieved. However, for $p > 4$, at least $s = p + 1$ stages are necessary. This order can actually be achieved with this number of stages only for $p = 5$ and $p = 6$. After that the required numbers of stages increases even more quickly. We summarize the situation as far as order eight, after which the exact value of s to achieve order p in unknown.

For a more detailed study of order conditions and other theoretical questions concerning explicit Runge–Kutta methods, see, for example, Refs. 1–3. Some of the mechanics of using the order conditions can be handled by the MATHEMATICA package NumericalMath`Butcher,` which is available in Version 2.0 and later versions.

Stiff problems

Many practical problems possess the disconcerting property that their solutions are extremely stable but numerical approximations are extremely unstable. An example of this can be found in the forced damped oscillator Eq. (4), where the period of the forcing term $2\pi/w$ is large compared with the time constant $2/a$ associated with the damping term. In this case, it is possible to obtain useful numerical results using an explicit Runge–Kutta method only when the value of h is small compared with the time constant. However, this restriction on h may well be an inappropriate restriction when viewed in terms of the physical significance of the result being computed. This will be illustrated in the case $a = 2000$, $b = 1000100$, and $w = 2\pi$, where the initial values and the value of c are consistent with an exact result of the form $z = \sin(2\pi x + \alpha)$ at x_0. The numerical results found in integrating from $x = 0$ to $x = 1$ are shown in Table IV. The value of "error" for each h is $(e_1^2 + e_2^2)^{1/2}$, where e_1 and e_2 are the errors in the computations of y^1 and y^2, respectively.

For $h \leqslant 1/400$, we observe the familiar pattern of decreasing errors, more or less in the ratio 16 to 1, as h is decreased in the ratio 2 to 1. However, there is a sudden jump in the error once h becomes large. The reason for this is that the linear part of the differential equation contains an auxiliary polynomial with zeros having large negative real

Table IV. Error behavior for a stiff problem.

h	$\frac{1}{100}$	$\frac{1}{200}$	$\frac{1}{400}$	$\frac{1}{800}$	$\frac{1}{1600}$	$\frac{1}{3200}$	$\frac{1}{6400}$
Error	1.6531×10^{23}	1.4853×10^{21}	5.9144×10^{-1}	2.7574×10^{-3}	1.1243×10^{-4}	5.7178×10^{-6}	3.2057×10^{-7}

parts. In fact, the zeros are $-1000 \pm 10i$. This means that, although the exact solution contains rapidly decreasing transient terms (which, with the initial values proposed for this example, make exactly zero contribution), the numerical approximations to these essentially negligible terms can be exceedingly significant. This phenomenon is known as "stiffness" and creates a difficulty of this sort whenever a numerical approximation is attempted with an explicit method. (In a subsequent issue, a column will be presented discussing stiff differential equations in detail.)

It can be shown that we retain stable behavior for the numerical approximations only if $h\lambda$ lies in the "stability region" for the method, where λ is any zero of the auxiliary polynomial. The stability region is defined as the part of the complex plane for which $|R(z)| \leqslant 1$, where $R(z)$, the so-called "stability polynomial," is defined to be

$$R(z) = 1 + zb^T(I - zA)^{-1}e, \qquad (14)$$

with $e = [1, 1, \ldots, 1]^T$. This can be illustrated in the special case of the differential equation

$$\frac{dy}{dx} = \lambda y, \qquad (15)$$

so that λ is the only zero of the auxiliary polynomial.

If $z = h\lambda$, the vector of stage values, $Y = [Y_1, Y_2, \ldots, Y_s]^T$, and the result computed in the step, y_n, are related to y_{n-1} by

$$Y = y_{n-1}e + zAY, \qquad (16)$$

$$y_n = y_{n-1} + zb^TY. \qquad (17)$$

Solve for Y from Eq. (16) and substitute into Eq. (17) to find

$$y_n = R(z)y_{n-1}, \qquad (18$$

where $R(z)$ is given by Eq. (14).

For the particular method we have used, the stabilit region is that part of the complex plane containing each z for which

$$|R(z)| = \left|1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}\right| \leqslant 1. \qquad (19$$

Since negative real parts of λ are associated with damped terms in the exact solution, we want all such terms to be damped in the numerical approximation. This means that the only really satisfactory methods for stiff problems are those for which the stability region contains the entire left half-plane; that is, $|R(z)| \leqslant 1$ whenever the complex number z has non-negative real part.

For explicit methods, the function $R(z)$ is a nonconstant polynomial and A stability is impossible. However, for implicit Runge–Kutta methods, which we will discuss in the next issue, $R(z)$ is a rational function, and A stability is not only possible but can be achieved for any order.

Further reading

1. J. C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge–Kutta and General Linear Methods* (Wiley, Chichester, 1987).
2. J. C. Butcher, J. Austral. Math. Soc. **3**, 185 (1963).
3. E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I, Nonstiff Problems,* 2nd ed. (Springer, Berlin, 1992).

RUNGE–KUTTA METHODS IN MODERN COMPUTATION, PART II: IMPLICIT METHODS AND RELATED APPLICATIONS

J. C. Butcher

Department Editor: William J. Thompson

Part I of this two-part article, which appeared in the Jul/Aug 1994 issue, p. 411, discussed fundamental concepts such as the order of Runge–Kutta methods, arborescences, the use of tableaux, stiffness, and stability. The explicit Runge–Kutta method was illustrated by means of the forced-damped-oscillator problem. Part II considers implicit Runge–Kutta methods, error estimation, implementation issues, and the solution of related problems: partial-differential and differential–algebraic equations.

We will now discuss the consequences within the Runge–Kutta approach of abandoning the restriction that each stage depends explicitly on previous stages. That is, we consider "implicit" Runge–Kutta methods. As we have remarked, explicit methods can never be A stable. Our hope will be that, within the wider implicit Runge–Kutta class, there do exist methods that possess this property and that they are thus suitable for the solution of stiff problems.

A well-known example of an implicit Runge–Kutta method is given by the tableau

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \quad . \tag{1}$$

Note the nonzero elements on and above the main diagonal: these are what makes the method implicit, because they indicate a dependence of a given stage value on derivatives evaluated at this stage and possibly at later stages.

The method (1) can be verified to have order four. Thus, it would seem to be more efficient than an explicit method would have been, because an explicit method would require four stages to achieve this order. However, the improved order is at an enormous cost. For an N-dimensional problem, we would need to solve an algebraic equation system in $2N$ unknowns to advance the numerical approximation through each individual time step. If the stage values are given by $\mathbf{Y}_1$ and $\mathbf{Y}_2$, these equations take the form

$$\begin{aligned} \mathbf{Y}_1 &= \mathbf{y}_{n-1} + h[a_{11}\mathbf{f}(\mathbf{Y}_1) + a_{12}\mathbf{f}(\mathbf{Y}_2)], \\ \mathbf{Y}_2 &= \mathbf{y}_{n-1} + h[a_{21}\mathbf{f}(\mathbf{Y}_1) + a_{22}\mathbf{f}(\mathbf{Y}_2)]. \end{aligned} \tag{2}$$

If s, the number of stages, were any higher than two, so as to obtain higher order, the algebraic system to be solved would take the form

$$\begin{aligned} \mathbf{Y}_1 &= \mathbf{y}_{n-1} + h[a_{11}\mathbf{f}(\mathbf{Y}_1) + a_{12}\mathbf{f}(\mathbf{Y}_2) + \cdots + a_{1s}\mathbf{f}(\mathbf{Y}_s)], \\ \mathbf{Y}_2 &= \mathbf{y}_{n-1} + h[a_{21}\mathbf{f}(\mathbf{Y}_1) + a_{22}\mathbf{f}(\mathbf{Y}_2) + \cdots + a_{2s}\mathbf{f}(\mathbf{Y}_s)], \\ &\vdots \qquad \vdots \\ \mathbf{Y}_s &= \mathbf{y}_{n-1} + h[a_{s1}\mathbf{f}(\mathbf{Y}_1) + a_{s2}\mathbf{f}(\mathbf{Y}_2) + \cdots + a_{ss}\mathbf{f}(\mathbf{Y}_s)], \end{aligned} \tag{3}$$

and the complexity of this algebraic equation system would rise rapidly. In many cases, such as the solution of the initial-value problem formed by the space discretization of a time-dependent partial-differential equation, N is extremely large, and the added cost of multiplying the size of the system by s is prohibitive. Hence, we will look at alternatives to the completely implicit type of method that we have exemplified. However, before leaving this method we note that its stability function is

$$R(z) = \frac{1 + z/2 + z^2/12}{1 - z/2 + z^2/12}. \tag{4}$$

It is easy to see by elementary complex analysis that $|R(z)| \leq 1$, whenever the complex number z has nonpositive real part. In accordance with a definition introduced in the previous section, this means that the method is A stable. This desirable property is actually shared by all methods for which the order is $p = 2s$. There is a unique method with this order for each value of s, but (as pointed out above) they are expensive to use.

John Butcher is a professor of mathematics at The University of Auckland in Auckland, New Zealand. He has made numerous contributions to the theory of Runge–Kutta methods and other aspects of the numerical solution of ordinary differential equations, and is the author of a definitive monograph (see part I, Ref. 1) on Runge–Kutta and general linear methods. His current interests include the extension of singly implicit Runge–Kutta methods to the solution of differential–algebraic equations and the solution of ordinary differential equations in a parallel environment. The work in the present paper was supported by the New Zealand Foundation for Research, Science and Technology.

An attractive alternative to fully implicit methods is provided by methods for which the matrix A has the structure

$$A=\begin{bmatrix} \lambda & 0 & 0 & \cdots & 0 \\ a_{21} & \lambda & 0 & \cdots & 0 \\ a_{31} & a_{32} & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & a_{s3} & \cdots & \lambda \end{bmatrix}. \qquad (5)$$

These are known as diagonally implicit Runge–Kutta (or DIRK) methods and are useful for many problems. Note that instead of having to solve a system of sN equations in every time step, we need only solve s systems of N equations. However, for high orders, these methods become less and less appropriate for the solution of stiff problems. In particular, the accuracy with which individual stages can be approximated and compared with the overall accuracy rapidly deteriorates with increasing s, and this is known to have an adverse influence on the effectiveness of these methods.

Much of the benefit associated with DIRK methods is also available with what are known as singly implicit Runge–Kutta (SIRK) methods. The fact that A given by Eq. (5) is triangular and has each diagonal element equal to λ implies that its characteristic polynomial is

$$\det(wI-A)=(w-\lambda)^s. \qquad (6)$$

SIRK methods are defined so that A has a characteristic polynomial of the form (6), just as for DIRK methods, although they need not necessarily have a simple triangular structure.

Methods with this property always exist for which the individual stages, and the overall results have orders s. (That is, the stage order of the method, as well as the order itself are each equal to s). Many of these methods are A stable (or very close to being A stable) and also have other desirable computational properties. Although each stage value depends on the derivatives for all stages, a linear transformation introduced within the iterative process for solving the algebraic equations, enables the cost (at least for large problems) to be lowered to a level similar to that for DIRK methods. An example of an A-stable singly implicit method with $p=s=2$ is given by the tableau

$$\begin{array}{c|cc} 3-2\sqrt{2} & \frac{5}{4}-\frac{3\sqrt{2}}{4} & \frac{7}{4}-\frac{5\sqrt{2}}{4} \\ 1 & \frac{1}{4}+\frac{\sqrt{2}}{4} & \frac{3}{4}-\frac{\sqrt{2}}{4} \\ \hline & \frac{1}{4}+\frac{\sqrt{2}}{4} & \frac{3}{4}-\frac{\sqrt{2}}{4} \end{array}\ . \qquad (7)$$

Properties of SIRK methods and their efficient implementation are discussed in Refs. 1–4. We will also discuss the implementation of these methods below. Implicit Runge–Kutta methods in general are discussed in Refs. 5 and 6.

Error estimation

Although we have discussed Runge–Kutta methods for differential equations under the tacit assumption that the stepsize h never changes throughout the integration, it is often, indeed usual, that varying h is more efficient than holding it constant. This can be illustrated with the example of a comet moving in a very eccentric orbit. While the comet is close to the sun, it is necessary to use a relatively small value of h because the gravitational field is changing rapidly, and great inaccuracies would result if accelerations changed too much during any time step. On the other hand, at great distances from the sun, velocity and acceleration components are small, and it would be wasteful of computer resources to carry out many steps with small h. A reasonable aim of differential equation software development is to leave such choices to automatic selection. Hence, strategies must be found for controlling h according to the value of computed quantities. The most important of such computed quantities would be estimates made during the integration of the error contributed in each step. The aim of the automatic selection scheme would be to control h so that this local error estimate never gets too large or unnecessarily small.

For explicit Runge–Kutta methods, the most popular means of estimating local truncation error has become the embedding of two methods into the same overall scheme. That is, schemes of the form

$$\begin{array}{c|cccccc} 0 & 0 & 0 & \cdots & 0 & 0 \\ c_2 & a_{21} & 0 & \cdots & 0 & 0 \\ c_3 & a_{31} & a_{32} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \\ \hline & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s \end{array} = \begin{array}{c|c} c & A \\ \hline & b^T \\ \hline & \hat{b}^T \end{array} \qquad (8)$$

are used in which

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \qquad (9)$$

is a method of some order p and

$$\begin{array}{c|c} c & A \\ \hline & \hat{b}^T \end{array} \qquad (10)$$

is a method of order $p+1$. Thus, the tableau (8) contains two methods in one; the first given by tableau (9) and the second by tableau (10), in which the coefficients $b_1, b_2, \ldots, b_s$ are replaced by $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_s$.

The idea is to use the lower-order method (9) to propagate the solution one further step and to use the difference between the results found by methods (9) and (10) to estimate the error in the step. Unfortunately, the cost of making the error estimation is considerable, so other procedures for carrying out the estimation have been suggested from time to time. For implicit Runge–Kutta methods for which the stage order is close to the order of the method, it is possible to provide an error estimate with little or no additional computation.

We conclude this discussion by presenting the tableau of a Runge–Kutta method with error estimate due to Fehlberg.[7] The order of the basic method is four and the higher-order method, providing a more accurate approximation for error-estimation purposes, is of order five;

$$\begin{array}{c|cccccc}
0 \\
\frac{2}{9} & \frac{2}{9} \\
\frac{1}{3} & \frac{1}{12} & \frac{1}{4} \\
\frac{3}{4} & \frac{69}{128} & -\frac{243}{128} & \frac{135}{64} \\
1 & -\frac{17}{12} & \frac{27}{4} & -\frac{27}{5} & \frac{16}{15} \\
\frac{5}{6} & \frac{65}{432} & -\frac{5}{16} & \frac{13}{16} & \frac{4}{27} & \frac{5}{144} \\
\hline
 & \frac{1}{9} & 0 & \frac{9}{20} & \frac{16}{45} & \frac{1}{12} \\
\hline
 & \frac{47}{450} & 0 & \frac{12}{25} & \frac{32}{225} & \frac{1}{30} & \frac{6}{25}
\end{array} \quad . \tag{11}$$

Further Runge–Kutta methods with error estimates are given, for example, in Refs. 8 and 9.

Implementation of Runge–Kutta methods

For implicit Runge–Kutta methods, there is an enormous cost associated with the nonlinear equation system arising in every step. For an s-stage method applied to an N-dimensional problem, the total number of unknowns to solve for, from an equal number of simultaneous equations, is sN. If a Newton-type method is used to solve these equations, the cost of a single iteration would be proportional to s^3N^3 for the factorization of the linearized system satisfied by the updates, together with a second cost of s^2N^2 for the substitutions required for the actual solution of this linear system. Because the Jacobian matrix for the problem does not usually change rapidly, it is often satisfactory to carry out the factorization only occasionally. It is then updated only when the speed of convergence deteriorates. Hence, in this discussion, we will take the Jacobian matrix for the function $\mathbf{f}$ as a constant matrix, which we write as $\mathbf{J}$.

Let $\mathbf{Y}_1,\mathbf{Y}_2,\ldots,\mathbf{Y}_s$ be approximations to the solutions of the system of equations given by Eq. (3), and let $\mathbf{W}_1$, $\mathbf{W}_2,\ldots,\mathbf{W}_s$ be the decrements in a single modified Newton step. That is, $\mathbf{Y}_k-\mathbf{W}_k$ is the improved approximation for stage k to be formed in the Newton step. In the updated step, assuming that we are entitled to approximate $\mathbf{f}$ by a linear function, the updated values of $\mathbf{f}(\mathbf{Y}_k)$ would be $\mathbf{f}(\mathbf{Y}_k-\mathbf{W}_k)$, or approximately $\mathbf{f}(\mathbf{Y}_k)-\mathbf{J}\mathbf{W}_k$.

Substituting into the equations to be satisfied, we find that

$$\begin{aligned}
&\mathbf{W}_k-h(a_{k1}\mathbf{J}\mathbf{W}_1+a_{k2}\mathbf{J}\mathbf{W}_2+\cdots+a_{ks}\mathbf{J}\mathbf{W}_s)\\
&\quad=\mathbf{Y}_k-\mathbf{y}_{n-1}-h[a_{k1}\mathbf{f}(\mathbf{Y}_1)+a_{k2}\mathbf{f}(\mathbf{Y}_2)+\cdots\\
&\qquad+a_{ks}\mathbf{f}(\mathbf{Y}_s)],\quad k=1,2,\ldots,s.
\end{aligned} \tag{12}$$

Because this is an $sN\times sN$ linear equation system, the remarks we have made concerning the computational cost obviously apply. Among the attempts that have been made to reduce this cost, we consider the transformation of this system to a sequence of smaller problems, by choosing a nonsingular matrix T such that $\bar{A}=T^{-1}AT$ is in Jordan canonical form. Suppose, for example, that all the eigenvalues of A are real and distinct. In this case, $\bar{A}$ would be a real diagonal matrix, say $\bar{A}=\operatorname{diag}(\lambda_1,\lambda_2,\ldots,\lambda_s)$. Let $\bar{\mathbf{Y}}_k$, $k=1,2,\ldots,s$ denote a sequence of linear combinations of stage vectors $\mathbf{Y}_k, k=1,2,\ldots,s$ that are formed using the matrix T^{-1}. The cost of forming $\bar{\mathbf{Y}}_1,\bar{\mathbf{Y}}_2,\ldots,\bar{\mathbf{Y}}_s$ from $\mathbf{Y}_1,\mathbf{Y}_2,\ldots,\mathbf{Y}_s$, or transforming in the reverse direction, is proportional to s^2N and is negligible for large N. Furthermore, the system of equations satisfied by similarly transformed decrement vectors $\bar{\mathbf{W}}_k$, $k=1,2,\ldots,s$ is now of the form

$$(I-h\lambda_k J)\bar{\mathbf{W}}_k=\mathbf{r}_k,\quad k=1,2,\ldots,s \tag{13}$$

and can be solved as s independent systems of N equations. The saving is greatest if the eigenvalues of A are all equal. Even though for methods defined in this way, $\bar{A}$ is a single Jordan block rather than a diagonal matrix, the cost is brought down to N^3 for the factorizations and sN^2 for the iterations. These are the singly implicit methods, already discussed, of which tableau (7) is an example.

In this brief discussion of the implementation of implicit Runge–Kutta methods, we have concentrated on the solution of the resulting system of algebraic equations. It should be said that many other implementation questions arise, and some of these questions relate also to explicit Runge–Kutta methods. For many applications, an *ad hoc* program can easily be put together by the user, but the use of libraries should not be overlooked. The IMSL and NAG libraries each contain significant sections on ordinary differential equations, and Runge–Kutta methods figure prominently. For an up-to-date review and comparison of many codes for solving both stiff and non-stiff problems, Refs. 6 and 10 are especially recommended.

Solution of related problems

Even though our discussion has been confined to initial-value differential equation problems, there are many closely related questions for which similar numerical techniques are available. We consider just two of these more general problem types: partial differential equations and differential–algebraic equations.

As an example of partial–differential equations, we consider the diffusion problem in one space dimension. To conform to standard notation, we will denote the time variable by t and the space variable by x

$$\frac{\partial u}{\partial t}=\nabla^2 u,\quad t>0,\quad 0<x<1, \tag{14}$$

with u specified for $t=0$ and $0<x<1$ and for $t>0$ and $x=0$ and $x=1$.

The solution for each t value can be regarded as a function on the interval given by $0<x<1$. The so-called "method of lines" is a technique for transforming an equation whose solution is such a function to one in which, for each t, the value of u is specified only at a finite mesh of points. For example, if equally spaced points are used, then we represent the solution for each t by a vector

$$y(t)=\begin{bmatrix} y^1(t)\\ y^2(t)\\ \vdots\\ y^N(t)\end{bmatrix}, \tag{15}$$

where

$$y^k(t)=u\left(\frac{k}{N+1},t\right). \tag{16}$$

Note that $y^0(t)$ and $y^{N+1}(t)$ represent boundary information at $x=0$ and $x=1$ and do not have ordinary differential equations associated with them.

To carry this method through, we need a suitable approximation to $\nabla^2 u$ at the mesh points. This is achieved by noting that, by Taylor's theorem,

$$u\left(x-\frac{1}{N+1}\right)-2u(x)+u\left(x+\frac{1}{N+1}\right)$$
$$=\frac{u''(x)}{(N+1)^2}+O(N^{-4}), \tag{17}$$

for a smooth function u. Hence, we can approximate u'' at meshpoint number k by $(N+1)^2(y_{k-1}-2y_k+y_{k+1})$ so that the method-of-lines discretization of Eq. (14) becomes

$$\frac{d}{dt}\begin{bmatrix} y^1(t) \\ y^2(t) \\ y^3(t) \\ \vdots \\ y^N(t) \end{bmatrix}$$
$$=(N+1)^2\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix}\begin{bmatrix} y^1(t) \\ y^2(t) \\ y^3(t) \\ \vdots \\ y^N(t) \end{bmatrix}$$
$$+(N+1)^2\begin{bmatrix} y^0(t) \\ 0 \\ 0 \\ \vdots \\ y^{N+1}(t) \end{bmatrix}. \tag{18}$$

This is an example of a stiff problem, because the matrix

$$(N+1)^2\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix} \tag{19}$$

has eigenvalues given by $-4(N+1)^2\sin^2[k\pi/2(N+1)]$, $k=1,2,...,N$, and the most negative of these is approximately $-4(N+1)^2+\pi^2$ compared with the least negative of approximately $-\pi^2$. It is the least negative of the eigenvalues, together with the behavior of the functions $y^0(t)$ and $y^{N+1}(t)$, that determine the physically observable components of the solution. Although the stiff nature of this problem makes it unsuitable for solution by an explicit Runge–Kutta method, A-stable methods such as those given by the tableaux (1) and (7) have no difficulties.

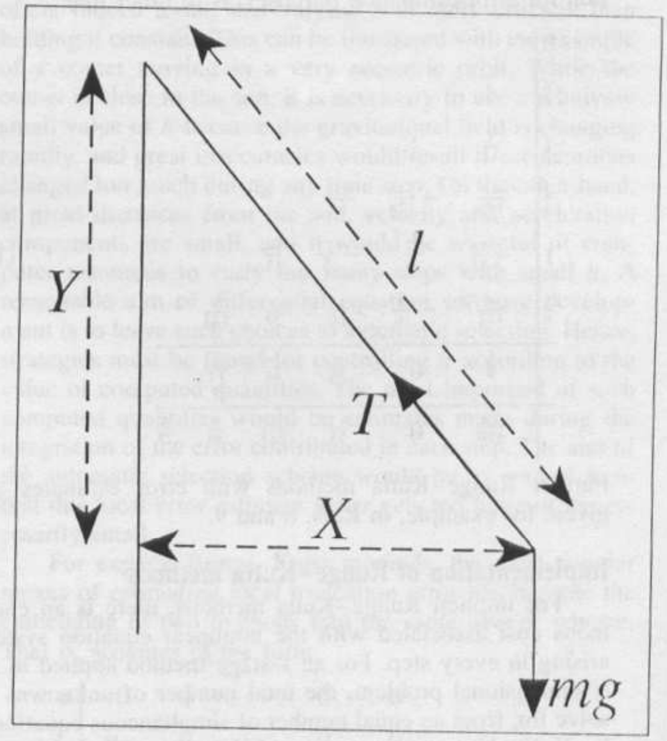

Figure 1. Variables for the simple pendulum.

As an example of a differential–algebraic equation, we consider an idealization of a standard type of dynamical problem, in which the motion is determined by a combination of the Newton equations of motion and of mechanical constraints. The problem is in fact the simple pendulum, in which a mass m is attached to an inflexible, inextensible, and weightless string of length l and can swing in a vertical plane. We will write g for the acceleration due to gravity. To model the equations of motion as a five-dimensional system, let X and Y denote position coordinates as shown in Fig. 1. U and V denote corresponding velocity components, and T denotes the tension in the string, which exactly balances the other forces so as to keep the length of the string constant.

From Fig. 1, we see that the motion is determined by the equations

$$\frac{dX}{dt}=U, \quad \frac{dY}{dt}=V,$$
$$m\frac{dU}{dt}=-\frac{TX}{\ell}, \quad m\frac{dV}{dt}=mg-\frac{TY}{\ell}, \tag{20}$$
$$X^2+Y^2=\ell^2,$$

where the first four are of standard differential-equation form. The fifth equation, corresponding to an algebraic constraint, is what makes this system differential–algebraic. Although equations of this type, arising in electrical network analysis as well as in constrained dynamical systems, can be converted to differential equations by repeated dif-

Table I. Two attempts to solve the differential–algebraic system Eq. (21).

	Gauss method [Eq. (1)]					Singly implicit method [Eq. (7)]				
n	y^1	y^2	y^3	y^4	y^5	y^1	y^2	y^3	y^4	y^5
5	0.862327	0.496171	−0.409217	−0.234717	27.001580	0.861220	0.508233	−0.005464	−0.001051	0.763124
	−0.003698	**−0.003829**	**−0.409217**	**−0.234717**	**26.501580**	**−0.004806**	**0.008233**	**−0.005464**	**−0.001051**	**0.263124**
10	0.865118	0.499032	−0.402271	−0.231839	101.37053	0.864485	0.502659	−0.001015	0.000421	0.569969
	−0.000907	**−0.000968**	**−0.402271**	**−0.231839**	**100.87053**	**−0.001541**	**0.002659**	**−0.001015**	**0.000421**	**0.069969**
20	0.865800	0.499757	−0.400454	−0.231098	398.662580	0.865595	0.500744	−0.000197	0.000173	0.517898
	−0.000226	**−0.000243**	**−0.400454**	**−0.231098**	**398.162580**	**−0.00430**	**0.000745**	**−0.000197**	**0.000173**	**0.017898**
40	0.865969	0.499939	−0.399995	−0.23091	1587.786124	0.865912	0.500196	−0.000041	0.000050	0.504511
	−0.000056	**−0.000061**	**−0.399995**	**−0.23091**	**1587.286124**	**−0.000113**	**0.000196**	**−0.000041**	**0.000050**	**0.004511**

ferentiation of the algebraic components, it is considered desirable for physical modeling to preserve the integrity of the constraints at all costs. The reduced pure differential-equation form leads to numerical approximations in which all components drift from the exact solution. This is to be expected, of course, but it is not reasonable to allow drift in the algebraic constraints. Hence, it is desirable to consider applying numerical methods to the differential–algebraic equation in its original form. We will investigate such a numerical solution using Runge–Kutta methods. For simplicity, we use a natural scaling of the problem so that m, l, and g are replaced by 1. This is achieved by the substitutions $t=x\surd(l/g)$, $X=ly^1$, $Y=ly^2$, $U=\surd(lg)y^3$, $V=\surd(lg)y^4$, $T=mgy^5$, so that the differential–algebraic system becomes

$$\frac{dy^1}{dx}=y^3,\quad \frac{dy^2}{dx}=y^4,$$
$$\frac{dy^3}{dx}=-y^1y^5,\quad \frac{dy^4}{dx}=1-y^2y^5, \tag{21}$$
$$(y^1)^2+(y^2)^2-1=0.$$

We will take the initial-value vector to be $y(0)=[0,1,1,0,2]^T$, corresponding to an angular amplitude of $\pi/3$. The solution with this amplitude can be shown, by evaluating an elliptic integral,[11] to have a period of $P=6.74300141925038$. In using a two-stage Runge–Kutta method to solve a problem of the form $\mathbf{F}(\mathbf{y}',\mathbf{y})=0$, we must solve in each step the algebraic equations

$$\begin{aligned}
&\mathbf{Y}_1=\mathbf{y}_{n-1}+h(a_{11}\mathbf{Z}_1+a_{12}\mathbf{Z}_2),\\
&\mathbf{Y}_2=\mathbf{y}_{n-1}+h(a_{21}\mathbf{Z}_1+a_{22}\mathbf{Z}_2),\\
&\mathbf{y}_n=\mathbf{y}_{n-1}+h(b_1\mathbf{Z}_1+b_2\mathbf{Z}_2),\\
&\mathbf{F}(Z_1,\mathbf{Y}_1)=0,\quad \mathbf{F}(\mathbf{Z}_2,\mathbf{Y}_2)=0,
\end{aligned} \tag{22}$$

where $\mathbf{Z}_1$ and $\mathbf{Z}_2$ have the same roles for this differential-algebraic equation system as $\mathbf{f}(\mathbf{Y}_1)$ and $\mathbf{f}(\mathbf{Y}_2)$ have for the ordinary differential-equation system [see Part I, Eq. (3)]. Note that explicit Runge–Kutta methods have no hope, because Eq. (22) is then inconsistent. We consider two choices of implicit method, those given by the tableaux (1) and (7).

We will use a time interval equal to $P/4$ and a stepsize equal to $h=P/4n$ for $n=5$, 10, 20, 40. The exact values at the output point are $[\surd 3/2,1/2,0,0,1/2]$. The results of these experiments are shown in Table I. Below each entry in the calculated results the error is shown in bold face.

A glance at this table shows the Gauss method to be a failure. However, the singly implicit method seems to give results quite acceptable for what is acknowledged to be a very difficult problem. For a more detailed study of differential-algebraic equations, see Refs. 6, 12, and 13.

Concluding remarks

Runge–Kutta methods have a long history, but they are still of immense interest. In the last 5 years alone, more than 600 papers referring to Runge–Kutta methods by name have appeared in print. They are used every day by physicists and other scientists to solve a wide variety of problems. In some cases they are chosen because of their simplicity, in other cases because of the need for the highly stable and accurate results they provide. Theoretical studies are frequently motivated by specific problem classes; for example, new stability and other structural requirements have been proposed and studied because of a perceived need to conserve in the computed solution such physical constraints as are known to be conserved by the exact solution. Other research is concerned with obtaining output at arbitrary points on the solution curve at little additional cost over what is needed to produce the step-by-step results discussed in this paper. The growing availability of parallel computing environments also provides new research challenges; Runge–Kutta methods undoubtedly have contributions to make to the numerical solution of problems so complex that serial computation is not feasible.

References

1. K. Burrage, BIT **18**, 22 (1978).
2. K. Burrage, J. C. Butcher, and F. H. Chipman, BIT **20**, 326 (1980).
3. J. C. Butcher, BIT **16**, 237 (1976).
4. J. C. Butcher, J. Assoc. Comput. Mach. **26**, 731 (1979).

5. J. C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge–Kutta and General Linear Methods* (Wiley, Chichester, 1987).
6. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential–Algebraic Problems* (Springer, Berlin, 1991).
7. E. Fehlberg, Computing **6**, 61 (1970).
8. J. R. Dormand and P. J. Prince, J. Comput. Appl. Math. **6**, 19 (1980).
9. J. H. Verner, SIAM J. Numer. Anal. **15**, 772 (1978).
10. E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I, Nonstiff Problems*, second ed. (Springer, Berlin, 1992).
11. *Handbook of Mathematical Functions*, edited by M. Abramowitz and I. A. Stegun (Dover, New York, 1965).
12. K. Brenan, S. Campbell, and L. Petzold, *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations* (North-Holland, Amsterdam, 1989).
13. R. März, Acta Numerica **1992**, 141 (1992).