ODE

Ordinary Differential Equations

1 Introduction

Historically, differential equations have originated in chemistry, physics, and engineering. More recently they have also arisen in models in medicine, biology, anthropology, and the like. In this chapter we restrict our attention to ordinary differential equations; a discussion of partial differential equations is a much more complicated issue and is given elsewhere in the book. We focus on initial value problems and present some of the more commonly used methods for solving such problems numerically. References are given where these methods have been implemented in quality software that is available on netlib using anonymous ftp.

1.1 Some Elements of the Theory

A *differential equation* is an equation involving an unknown function and one or more of its derivatives. The equation is an *ordinary differential equation* (ODE) if the unknown function depends on only one independent variable. Some examples of ODEs follow:

$$\frac{du}{dt} = F(t)G(u), \qquad \text{the growth equation;} \tag{1}$$

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = F(t), \qquad \text{the pendulum equation;} \tag{2}$$

$$\frac{d^2y}{dt^2} + \varepsilon(y^2+1)\frac{dy}{dt} + y = 0, \qquad \text{the van der Pol equation;} \tag{3}$$

$$L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{Q}{C} = E(t), \qquad \text{the LCR oscillator equation;} \tag{4}$$

$$\frac{dp}{dt} = -2a(t)p + \frac{b(t)^2}{u(t)}p^2 - v(t), \qquad \text{a Riccati equation.} \tag{5}$$

In (1–5) t is the independent variable; the dependent variables are u, θ, y, Q, and p, respectively.

In what follows we will frequently use the notation $\dot{y}$ to represent dy/dt, $\ddot{y}$ to represent d^2y/dt^2, $y^{(3)}$ to represent d^3y/dt^3, and, in general, $y^{(n)}$ to represent d^ny/dt^n.

The *order* of a differential equation is the order of the highest derivative appearing in the equation. Equations (2), (3), and (4) are second order equations and (1) and (5) are first order equations.

A *solution* of a general differential equation of the nth order,

$$f(t, y, \dot{y}, \ldots, y^{(n)}) = 0, \tag{6}$$

is a real-valued function $y(t)$ defined over some interval I having the following properties: 1) $y(t)$ and its first n derivatives exist for all t in I, so $y(t)$ and its first $n-1$ derivatives must be continuous in I, and 2) $y(t)$ satisfies the differential equation for all t in I.

Example 1 *Two differential equations and their solutions.*

a) The function,

$$y(t) = \frac{3}{4}(1+t^2) + \frac{5}{1+t^2},$$

is a solution to the differential equation

$$\frac{1}{t}\dot{y} + \frac{2}{1+t^2}y = 3.$$

b) The function

$$y(t) = c_1e^{-2t} + c_2e^{-3t} + \frac{1}{2}e^{2t}, \quad -\infty < t < \infty, \tag{7}$$

where c_1 and c_2 are arbitrary constants, is a solution to the differential equation

$$\ddot{y} + 5\dot{y} + 6y = 10e^{2t}.$$

In this case, $y(t)$ is also referred to as a *general* solution because all solutions to the differential equation can be represented in this form for appropriate choices of the constants c_1 and c_2. The function $y(t) = \frac{1}{2}e^{2t}$ is a *particular* solution because it contains no arbitrary constants.

With a differential equation, we can associate *initial conditions* or *boundary conditions*, auxiliary conditions on the unknown function and its derivatives. If these conditions are specified at a single value of the independent variable, they are referred to as initial conditions and the combination of the differential equation and an appropriate number of initial conditions is called an *initial value problem* (IVP). If these conditions are specified at more than one value of the independent variable, they are referred to as boundary conditions and the combination of the differential equation and the boundary conditions is called a *boundary value problem* (BVP).

Example 2 *Two examples of IVPs.*

a) The logistic equation,

$$\dot{p} = ap - bp^2,$$

with initial condition $p(t_0) = p_0$; for $p_0 = 10$ the solution is

$$p(t) = \frac{10a}{10b + (a - 10b)e^{-a(t-t_0)}}.$$

b) The mass-spring system equation,

$$\ddot{x} + (a/m)\dot{x} + (k/m)x = g + (F(t)/m),$$

with the initial conditions $x(0) = x_0$, $\dot{x}(0) = v_0$; for $m = 10$, $k = 140$, $a = 90$, $F(t) = 5\sin(t)$, $x_0 = 0$, $v_0 = -1$, the solution is

$$x(t) = \frac{1}{500}(-90e^{-2t} + 99e^{-7t} + 13\sin t - 9\cos t).$$

Example 3 *Two examples of BVPs.*

a) The differential equation,

$$\ddot{y} + 9y = \sin t,$$

with the boundary conditions $y(0) = 1$, $\dot{y}(2\pi) = -1$; the solution $y(t) = \frac{1}{8}\sin t + \cos 3t + \sin 3t$.
b) The differential equation,

$$\ddot{y} + \pi^2 y = 0,$$

with boundary conditions $y(0) = 2, y(1) = -2$; the solution is $y(t) = 2\cos\pi t + c\sin\pi t$, for c an arbitrary constant.

An nth-order differential equation is said to be *linear* if it can be written in the form

$$a_n(t)y^{(n)} + a_{n-1}(t)y^{(n-1)} + \ldots + a_1\dot{y} + a_0(t)y = h(t). \tag{8}$$

A *nonlinear* differential equation is simply one that is not linear. As examples, (4) is linear while (2), (3), and (5) are nonlinear. Equation (1) is linear when $G(u)$ is a linear function of u; otherwise, it is nonlinear. Differential equations arising from first principle models are generally nonlinear. Nonlinear equations do not usually yield to analytical approaches and computational methods are called for.

Linear equations constitute a highly important class of differential equations in physics and engineering and are used in idealized models of such phenomena as mechanical vibrations, electrical circuits, planetary motions, etc. An important property of linear equations is that of *superposition*: To illustrate the superposition principle, consider the following IVP:

$$\ddot{y} = g(t), \qquad y(0) = \alpha,\ \dot{y}(0) = \beta. \tag{9}$$

The data for this problem are $\{g(t), \alpha, \beta\}$. If $y_1(t)$ is a solution with data $\{g_1, \alpha_1, \beta_1\}$, and $y_2(t)$ is a solution with data $\{g_2, \alpha_2, \beta_2\}$, then the principle states that $c_1y_1(t) + c_2y_2(t)$ is a solution for the data $\{c_1g_1 + c_2g_2, c_1\alpha_1 + c_2\alpha_2, c_1\beta_1 + c_2\beta_2\}$. This idea extends readily to nth order differential equations. In practice, superposition permits us to decompose a problem with complicated data into simpler parts, to solve each problem separately, and then to combine these solutions to find the solution to the original problem.

Example 4 *An illustration of superposition.*

To solve the IVP,

$$\ddot{y} + 6\dot{y} + 9y = 4e^{-3t},$$
$$y(0) = 1,\ \dot{y}(0) = 0,$$

we solve the following two problems:

$$\ddot{y} + 6\dot{y} + 9y = 0,$$
$$y(0) = 1,\ \dot{y}(0) = 0,$$

a homogeneous equation with the original initial conditions, and

$$\ddot{y} + 6\dot{y} + 9y = 4e^{-3t},$$
$$y(0) = 0,\ \dot{y}(0) = 0,$$

an inhomogeneous equation with zero initial conditions. The solution to the first problem is

$$y_1(t) \quad = \quad \frac{1}{4}e^{-3t} + \frac{3}{4}te^{-3t},$$

and the second is

$$y_2(t) = 2t^2e^{-3t}.$$

So, the solution to the original problem is the sum

$$y(t) = y_1(t) + y_2(t) = \tfrac{1}{4}e^{-3t} + \tfrac{3}{4}te^{-3t} + 2t^2e^{-3t}.$$

A solution, $y(t)$, of a differential equation is said to be *stable* if any other solution whose initial data is sufficiently close to that of $y(t)$ remains in a "tube" enclosing $y(t)$; if the solution is not stable, it is said to be *unstable*. If the diameter of the tube approaches zero as t becomes large, then $y(t)$ is said to be *asymptotically stable*.

In elementary treatments of differential equations it is assumed that the initial value problem has a unique solution that exists throughout the interval of interest and which can be obtained by analytical techniques. However, many of the differential equations encountered in practice cannot be solve explicitly, so we are led to methods for obtaining approximations to solutions. Such solutions are usually called *numerical* solutions. Matters are also complicated by the fact that solutions can fail to exist over the desired interval of interest. Even more troublesome are problems with more than one solution.

Example 5 *Some examples of difficulties in solving IVPs.*

a) The differential equation,

$$\dot{y} = e^{t^2},$$

does not have a solution that can be expressed in terms of elementary functions.
b) The IVP,

$$\dot{y} = 1 + y^2,\ y(0) = 0,$$

has the solution $y(t) = \tan t$ which exists on the interval $0 \le t \le 1$ but does not exist on the interval $0 \le t \le \pi/2$.
c) The IVP,

$$\dot{y} = \sqrt{|1 - y^2|},\ y(0) = 1,$$

does not have a unique solution. In fact, it is not difficult to show that:

1) $y(t) = 1$ is a solution on any interval containing $t = 0$;
2) $y(t) = \cosh(t)$ is a solution on any interval $0 \le t \le b$ for any $b > 0$;
3) $y(t) = \cos(t)$ is a solution on $-\pi \le t \le 0$ and this is the largest such interval on which $\cos(t)$ is a solution.

---

2 Systems of Differential Equations

Suppose we want to solve the system of first order ODEs,

$$\dot{\mathbf{Y}} \ = \ \mathbf{F}(t, \mathbf{Y}), \qquad \mathbf{Y}(a) = \mathbf{A}. \tag{10}$$

Here $\mathbf{Y}$ and $\mathbf{A}$ are n-vectors and $\mathbf{F}$ is a nonlinear vector-valued function on $R \times R^n$:

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_n \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_n \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} F_1(t, Y_1, Y_2, \ldots, Y_n) \\ F_2(t, Y_1, Y_2, \ldots, Y_n) \\ \cdots \\ F_n(t, Y_1, Y_2, \ldots, Y_n) \end{bmatrix}. \tag{11}$$

To insure existence and uniqueness of a solution to (10) and hence establish effective numerical procedures, we require that $\mathbf{F}(t, \mathbf{Y})$ and $\partial\mathbf{F}/\partial\mathbf{Y}$ be continuous in the box $\mathbf{B}: |t-a| \leq \alpha$, $\|\mathbf{Y} - A\| \leq \beta$ where α and β are positive numbers and $\|\ \|$ is the vector Euclidean norm. If $\|\mathbf{F}(t, \mathbf{Y})\| \leq M$ for all $(t, \mathbf{Y})$ in $\mathbf{B}$ and if h is the smaller of α and β/M, then the IVP (10) has a unique solution for $|t-a| \leq h$. Weaker conditions do exist, but these will suffice for our purposes. For a more detailed discussion of existence and uniqueness issues, consult any good text on differential equations, such as [1, 2, 3].

Most computer codes for solving IVPs accept problems in the form (10). However, problems often arise in different forms. For example, one is often interested in solving the second order equation,

$$\ddot{y} = g(t, y, \dot{y}), \tag{12}$$

with initial conditions $y(a) = A_1$, $\dot{y}(a) = A_2$. Here there is one unknown function $y(t)$ from which we can, *in principle*, obtain $\dot{y}(t)$. A new problem can be formed that involves two unknown quantities, $Y_1(t)$ and $Y_2(t)$, but which is in the form (10). The idea is to identify *independently* $y(t)$ by $Y_1(t)$ and $\dot{y}(t)$ by $Y_2(t)$. Some manipulation then shows that $Y_1(t)$ and $Y_2(t)$ satisfy the system,

$$\dot{Y}_1 = Y_2, \tag{13}$$

$$\dot{Y}_2 = g(t, Y_1, Y_2), \tag{14}$$

with

$$Y_1(a) = A_1, \ Y_2(a) \ = \ A_2. \tag{15}$$

Then $y(t) = Y_1(t)$ is the solution of the original problem. This can be put into the form (10) by defining

$$\mathbf{Y} = \begin{bmatrix} y \\ \dot{y} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} Y_2 \\ g(t, Y_1, Y_2) \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}.$$

More generally, an nth order equation in one unknown,

$$\begin{aligned} y^{(n)} &= g(t, y, \dot{y}, \ldots, y^{(n-1)}), \\ y(t_0) &= A_1,\ \dot{y}(t_0) = A_2, \ldots, y^{(n-1)} = A_n, \end{aligned} \tag{16}$$

can be put into the form (10) via

$$\mathbf{Y} = \begin{bmatrix} y \\ \dot{y} \\ \ldots \\ y^{(n-2)} \\ y^{(n-1)} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} Y_2 \\ Y_3 \\ \ldots \\ Y_n \\ g(t, Y_1, Y_2, \ldots, Y_n) \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} A_1 \\ A_2 \\ \ldots \\ A_n \end{bmatrix}.$$

The general procedure here is to introduce unknowns to handle derivatives up to one less than the highest appearing in the equation.

Example 6 *Conversion of a higher-order system to an equivalent system of first order equations.*

Consider the simplified equations of planetary motion,

$$\begin{aligned} \ddot{r} - r(\dot{\theta}) &= -\frac{K}{r^2}, \\ r\ddot{\theta} + 2\dot{\theta}\dot{r} &= 0, \end{aligned}$$

with the initial conditions $r(0) = r_0$, $\dot{r}(0) = \dot{r}_0$, $\theta(0) = \theta_0$, and $\dot{\theta}(0) = \dot{\theta}_0$. Letting $Y_1 = r$, $Y_2 = \dot{r}$, $Y_3 = \theta$, and $Y_4 = \dot{\theta}$, we derive the equivalent first order system,

$$\begin{aligned} \dot{Y}_1 &= Y_2, \\ \dot{Y}_2 &= Y_1 Y_4 - \frac{K}{Y_1^2}, \\ \dot{Y}_3 &= Y_4, \\ \dot{Y}_4 &= -\frac{2Y_4 Y_2}{Y_1}, \end{aligned}$$

with initial conditions $Y_1(0) = r_0$, $Y_2(0) = \dot{r}_0$, $Y_3(0) = \theta_0$, and $Y_4(0) = \dot{\theta}_0$.

This is the usual way to go from a higher order system to a system of first order equations, but there are many other ways to do it; see Exercise 1.3 and Example 7.

Example 7 *A nonstandard way to convert a second-order ODE to an equivalent system of first order ODEs.*

The following problem arises in the study of nonlinear mechanics:

$$\ddot{y} + g(y)\dot{y} + y = 0.$$

The change of variables, $Y_1 = y, Y_2 = \dot{y}$, yields the system

$$\begin{aligned}\dot{Y}_1 &= Y_2,\\ \dot{Y}_2 &= -g(Y_1)Y_2 - Y_1.\end{aligned}$$

However, in many problems the function $g(x)$ is simple in form, but only smooth in pieces. If we have an analytical expression for an indefinite integral $G(x)$ of the function $g(x)$, that is $d/dx(G(x)) = g(x)$, then we note that

$$\ddot{y} + g(y(t))\dot{y}(t) + y(t) = \frac{d}{dt}[\dot{y}(t)G(y(t))] + y(t).$$

So we use the change of variables suggested by Lienard,

$$Y_1(t) = y(t), \qquad Y_2(t) = \dot{y}(t) + G(y(t)),$$

to get the system

$$\begin{aligned}\dot{Y}_1 &= Y_2 - G(Y_1),\\ \dot{Y}_2 &= -Y_1.\end{aligned}$$

This system has smoother functions than the standard system making it easier to solve numerically.

See exercise set 1.

2.1 Numerical Solution Methods

We begin with numerical methods for solving a scalar version of (10), i.e., the case for $n = 1$:

$$\dot{y} = f(t, y), \qquad y(a) = A. \tag{17}$$

The methods we develop for solving (17) can easily be extended to systems of first order differential equations and to higher order differential equations. The methods are referred to as discrete variable methods and generate a sequence of approximate values for $y(t), y_1, y_2, y_3, \cdots$, at points $t_1, t_2, t_3, \cdots$. No attempt is made to approximate the exact solution, $y(t)$, over a continuous range of the independent variable t. In our development, we will assume a constant spacing h between t points. In realistic implementations of these methods, however, h is chosen to satisfy a user-specified accuracy request. The expression $y(t_i)$ will always be used to denote the solution to (17) at $t = t_i$, and y_i will always be used for an approximation to $y(t_i)$.

Errors enter into the numerical solution of IVPs from two sources. The first is *discretization error* and depends on the method being used. The second is *computational error* which includes such things as *roundoff error*, the error in evaluating implicit formulas, etc. In general, roundoff error can be controlled by carrying enough significant figures in the computation. The control of other computational errors again depends on the method being used.

There are two measures of discretization error commonly used in discussing the accuracy of numerical methods for solving IVPs. The first is *true* or *global* error. For any $t = t_{i+1}$, global error is simply the difference between the true solution and our numerical approximation to it:

$$e_{i+1} = y(t_{i+1}) - y_{i+1}. \tag{18}$$

Even though this is the error in which we are usually interested, it is a relatively difficult and expensive to estimate. The other measure of error is *local error*. It is the error incurred in taking a single step using a numerical method. If we let $u(t)$ be the solution to the IVP,

$$\dot{u} = f(t, u), \qquad u(t_i) = y_i, \tag{19}$$

then the local error at $t = t_{i+1}$ is given by

$$d_{i+1} = u(t_{i+1}) - y_{i+1}. \tag{20}$$

Most codes for solving IVPs estimate the local error at each step and attempt to adjust h accordingly. Control of local error controls global error indirectly; this, of course, depends on the stability of the problem itself. Most problems are at least moderately stable, and the global error is comparable to the error tolerance. Also, the cost of estimating global error is twice or more the cost of the integration itself.

Local and global errors at $t = t_{i+1}$ are illustrated graphically in Figure 1.

2.2 One-Step Methods

A differential equation has no "memory". That is, the values of $y(t)$ for t before t_i do not directly affect the values of $y(t)$ for t after t_i. Some numerical methods have memory, and some do not. We shall first describe a class of methods known as *one-step* methods. They have no memory; given y_i there is a recipe for y_{i+1} that depends only on information at t_i.

Suppose we want to approximate the solution to (17) on the interval $[a, b]$. Let the t points be equally spaced; so for some positive integer n and $h = (b-a)/n$, $t_i = a + ih$, $i = 0, 1, \ldots, n$. If $a < b$, h is positive and we are integrating forward; if $a > b$, h is negative

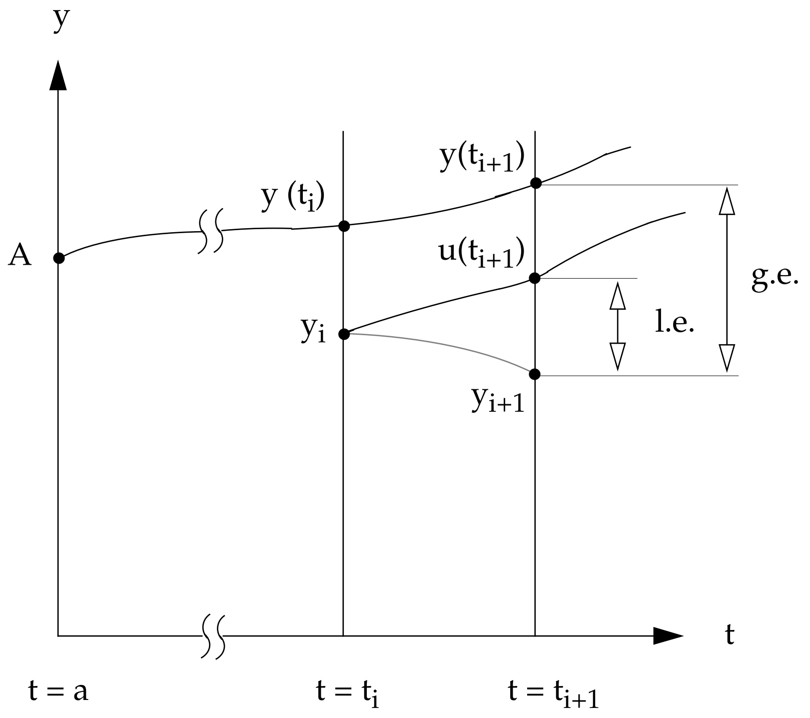

Figure 1: Local error (l.e.) and global error (g.e.) at $t = t_{i+1}$.

and we are integrating backwards. The latter case could occur if we were solving for the initial point of a solution curve given the terminal point. A general one-step method can then be written in the form

$$y_{i+1} = y_i + h\Delta(t_i, y_i), \qquad y_0 = y(t_0), \tag{21}$$

where Δ is a function that characterizes our method. We seek accurate algorithms of the form (21). By this we mean algorithms for which the true solution, $y(t)$, almost satisfies (21), i.e.,

$$y(t_{i+1}) = y(t_i) + h\Delta(t_i, y(t_i)) + h\tau_i \tag{22}$$

with τ_i "small." The quantity $h\tau_i$ is called the *local (truncation) error* of the method. The method (21) is said to be of *order* p if for all $t_i, a \leq t_i \leq b$, and for all sufficiently small h, there are constants C and p such that

$$|\tau_i| = Ch^p. \tag{23}$$

This can be interpreted as meaning that $|\tau_i|$ goes to zero no slower than Ch^p. Hereafter, we shall write terms like this as $\tau_i = O(h^p)$. The constant C depends, in general, on the solution $y(t)$, its derivatives, and the length of the interval over which the solution is to be found, but is independent of h.

Note that, the order of our method is p even though the order of the local (truncation) error is $p+1$, because these errors tend to accumulate as the integration proceeds. The order of a method may be viewed as a measure of how fast the error in the computed solution goes to zero at a fixed point t as more and more steps are taken, i.e., as h approaches zero. Our goal is to find functions Δ that are inexpensive to evaluate, yet of as high order p as possible. In what follows, different Δ functions are displayed, giving rise to the Taylor series methods and the Runge-Kutta methods.

2.2.1 Taylor Series Methods

Perhaps the simplest one-step methods of order p are based on Taylor series expansion of the solution $y(t)$. If $y^{(p+1)}(t)$ is continuous on $[a, b]$, then Taylor's formula gives

$$y(t_{i+1}) = y(t_i) + h\left[\dot{y}(t_i) + \ldots + y^{(p)}(t_i)\frac{h^{p-1}}{p!}\right] + y^{(p+1)}(\zeta_i)\frac{h^{p+1}}{(p+1)!}, \tag{24}$$

where $t_i \leq \zeta_i \leq t_{i+1}$. The continuity of $y^{(p+1)}(t)$ implies that it is bounded on $[a, b]$ and so

$$y^{p+1}(\zeta_i)\frac{h^{p+1}}{(p+1)!} = O(h^{p+1}) = hO(h^p). \tag{25}$$

Using the fact that $\dot{y} = f(t, y)$, (24) can be written in the form

$$y(t_{i+1}) = y(t_i) + h\left[f(t_i, y(t_i)) + \ldots + f^{(p-1)}(t_i, y(t_i))\frac{h^{(p-1)}}{p!}\right] + hO(h^p), \tag{26}$$

where the total derivatives of f are defined recursively by

$$\begin{aligned} f^{(1)}(t, y) &= f_t(t, y) + f_y(t, y)f(t, y), \\ f^{(k)}(t, y) &= f_t^{(k-1)}(t, y) + f_y^{(k-1)}(t, y)f(t, y), \quad k = 2, 3, \cdots. \end{aligned}$$

Comparison of (22) and (26) shows that to obtain a method of order p, we can let

$$\Delta(t, y(t_i)) = f(t_i, y(t_i)) + \ldots + f^{(p-1)}(t_i, y(t_i))\frac{h^{p-1}}{p!}. \tag{27}$$

This choice leads to a family of methods known as the Taylor series methods, given in the following algorithm.

Algorithm 2.1 *Taylor series algorithm*

To obtain an approximate solution of order p to the IVP (17) on $[a, b]$, let $h = (b-a)/n$ and generate the sequences

$$\begin{aligned} y_{i+1} &= y_i + h\left[f(t_i, y_i) + \ldots + f^{(p-1)}(t_i, y_i)\frac{h^{p-1}}{p!}\right], \\ t_{i+1} &= t_i + h, \quad i = 0, 1, \ldots, n-1, \end{aligned} \tag{28}$$

where $t_0 = a$, and $y_0 = A$.

Example 8 *The Euler method.*

The Taylor method of order $p = 1$ is known as Euler's method:

$$\begin{aligned} y_{i+1} &= y_i + hf(t_i, y_i), \\ t_{i+1} &= t_i + h. \end{aligned} \tag{29}$$

To illustrate it, we approximate the solution to the IVP,

$$\dot{y} = y,\ y(0) = 1,$$

at $t = 1.0$ with step $h = 0.25$. Here $f(t, y) = y$, so (29) simplifies to

$$\begin{aligned} y_{i+1} &= y_i + 0.25y_i, \\ t_{i+1} &= t_i + h. \end{aligned}$$

Starting with $t_0 = 0.0$ and $y_0 = 1.0$, we compute, truncating results to four decimal places:

$$\begin{aligned} y_1 &= y_0 + 0.25y_0 = (1.0) + (0.25)(1.0) = 1.25, \\ t_1 &= t_0 + h = 0.0 + (0.25) = 0.25; \\[1em] y_2 &= y_1 + 0.25y_1 = (1.25) + (0.25)(1.25) = 1.5625, \\ t_2 &= t_1 + h = 0.25 + (0.25) = 0.50; \\[1em] y_3 &= (1.5625) + (0.25)(1.5625) = 1.9531, \\ t_3 &= t_2 + h = 0.50 + (0.25) = 0.75; \\[1em] y_4 &= (1.9531) + (0.25)(1.9531) = 2.4414, \\ t_4 &= t_3 + h = 0.75 + (0.25) = 1.00. \end{aligned}$$

The exact solution is $y(t) = e^t$, so $y(1) = 2.7183$, correct to four decimal places, and the magnitude of the true or global error in our approximation is $|y_4 - y(1)| = 0.2768$. The approximate and exact solutions are represented graphically below in Figure 2, where the approximating values y_1, y_2, y_3, and y_4 have been joined by straight line segments.

If we are willing to do more work, we can obtain the results given in Table 1.
We note from the last column of Table 1 that $|y_n - e| = O(h)$; in fact, for $n \geq 16$, $|y_n - e| = 1.3h$. This is consistent with the fact that Euler's method is of order $p = 1$.

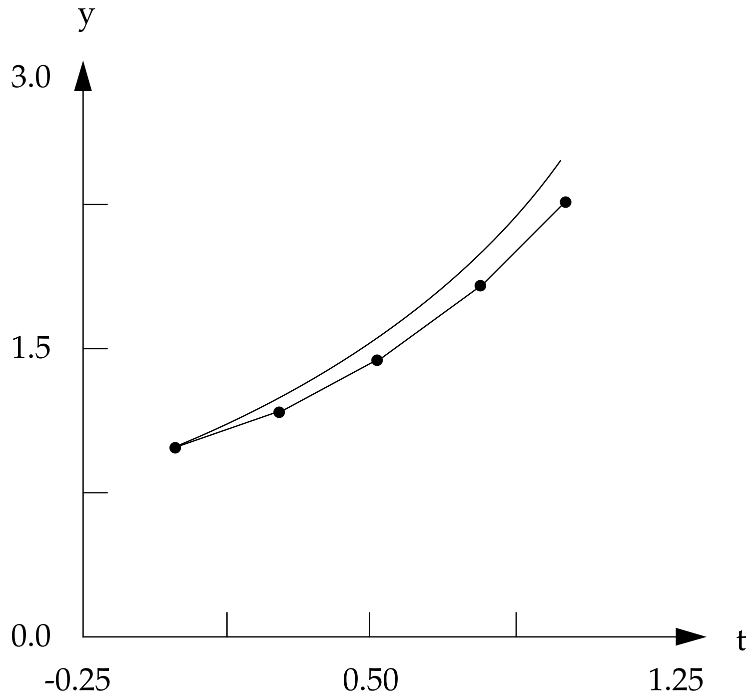

Figure 2: Plot of the Euler solution and exact solution to $\dot{y} = y$, $y(0) = 1$.

n	$h = 1/n$	y_n	$\|y_n - e\|$	$(y_n - e)/h$
8	0.12500e+00	2.56578	0.15e+00	0.12e+01
16	0.62500e-01	2.63793	0.80e-01	0.13e+01
32	0.31250e-01	2.67699	0.41e-01	0.13e+01
64	0.15625e-01	2.69735	0.21e-01	0.13e+01
128	0.78125e-02	2.70774	0.11e-01	0.13e+01

Table 1: Approximate solution to $\dot{y} = y$, $y(0) = 1$ at $t = 1$ for small h using Euler's method.

n	$h = 1/n$	y_n	$\|y_n - e\|$	$(y_n - e)/h^2$
4	0.25000e+00	2.69486	0.23e-01	0.37e+00
8	0.12500e+00	2.71184	0.64e-02	0.41e+00
16	0.62500e-01	2.71659	0.17e-02	0.43e+00
32	0.31250e-01	2.71785	0.43e-03	0.44e+00
64	0.15625e-01	2.71817	0.11e-03	0.45e+00
128	0.78125e-02	2.71825	0.27e-04	0.45e+00

Table 2: Approximate solution to $\dot{y} = y$, $y(0) = 1$ at $t = 1$ for small h using the Taylor series algorithm of order 2.

If we repeat the above calculations using a Taylor method of order 2, we obtain

$$\begin{aligned} y_{i+1} &= y_i + h(y_i + \frac{h}{2} y_i), \\ t_{i+1} &= t_i + h, \end{aligned}$$

and the results in Table 2. From Table 2, we see that for $n \geq 16$, $|y_n - e| = 0.45h^2$.

What would happen if we were to increase the order even more? For $p = 3$ and $p = 4$ the formulas become

$$\begin{aligned} \text{For } p = 3, \quad y_{i+1} &= y_i(1 + h + \frac{h^2}{2} + \frac{h^3}{6}), \\ \text{For } p = 4, \quad y_{i+1} &= y_i(1 + h + \frac{h^2}{2} + \frac{h^3}{6} + \frac{h^4}{24}), \end{aligned}$$

and we obtain using $h = 0.25$ the results that appear in Table 3. Note that the same order of accuracy is obtained for $p = 4$ with $h = 0.25$ as for $p = 2$ and $h = 7.8125 \times 10^{-3}$. Of course, higher total derivatives $f^{(n)}$ are easy to compute for $f(t, y) = y$.

Taylor series methods can be quite effective if the total derivatives of f are not too difficult to evaluate. Software packages are available that perform exact differentiation, facilitating their use (ADIFOR, MAPLE, MATHEMATICA, etc.). However, most of today's software packages for solving IVPs do not employ Taylor series methods.

See exercise set 2.

2.2.2 Runge-Kutta Methods

Runge-Kutta methods are designed to approximate Taylor series methods, but have the advantage of not requiring explicit evaluations of the derivatives of $f(t, y)$. The basic idea is to use a linear combination of values of $f(t, y)$ to approximate $y(t)$. This linear combination is matched up as closely as possibly with a Taylor series for $y(t)$ to obtain methods of the highest possible order p. Euler's method is an example using one function evaluation.

order	y_4	$\|y_4 - e\|$
3	2.71683	0.14e-02
4	2.71821	0.72e-04

Table 3: Approximate solution to $\dot{y} = y$, $y(0) = 1$ at $t = 1$ for $h = 0.25$ using the Taylor series algorithms of orders 3 and 4.

We illustrate the development of Runge-Kutta formulas by deriving a method using two evaluations of $f(t, y)$ per step; the technique employed in the derivation extends easily to the development of all Runge-Kutta type formulas. Given values t_i, y_i, choose values $\hat{t}_i$, $\hat{y}_i$ and constants α_1, α_2 so as to match

$$y_{i+1} = y_i + h\left[\alpha_1 f(t_i, y_i) + \alpha_2 f(\hat{t}_i, \hat{y}_i)\right] \tag{30}$$

with the Taylor expansion,

$$y(t_{i+1}) = y_i + \left[f(t_i, y_i) + f^{(1)}(t_i, y_i)\frac{h}{2} + f^{(2)}(t_i, y_i)\frac{h^2}{6}\cdots\right], \tag{31}$$

as closely as possible. In what follows all arguments of f and its derivatives will be suppressed when they are evaluated at (t_i, y_i). It will also be convenient to express $\hat{t}_i$, $\hat{y}_i$ as

$$\begin{aligned}
\hat{t}_i &= t_i + h\beta_i, \\
\hat{y}_i &= y_i + \beta_2 h f(t_i, y_i).
\end{aligned}$$

So the object is to match

$$\begin{aligned}
R &= \alpha_1 f + \alpha_2 f(t_i + \beta_1 h, y_i + \beta_2 h f) \\
&= (\alpha_1 + \alpha_2) f + \alpha_2 h(\beta_2 f f_y + \beta_1 f_t) + \frac{\alpha_2 h^2}{2}(\beta_2^2 f^2 f_{yy} + 2\beta_1\beta_2 f f_{ty} + \beta_1^2 f_{tt}) \\
&\quad + O(h^3)
\end{aligned} \tag{32}$$

with the Taylor expansion

$$\begin{aligned}
T &= f + \frac{h}{2} f^{(1)} + \frac{h^2}{6} f^{(2)} + O(h^3) \\
&= f + \frac{h}{2}(f f_y + f_t) + \frac{h^2}{6}(f^2 f_{yy} + 2 f f_{ty} + f_{tt} + f_t f_y + f f_y^2) + O(h^3).
\end{aligned} \tag{33}$$

Equating coefficients of like powers of h in the above expressions for R and T, we are able to obtain agreement in terms involving h^0 and h^1:

$$\begin{aligned}
h^0: &\quad \alpha + \alpha_2 = 1, \\
h^1: &\quad \alpha_2\beta_2 = \alpha_2\beta_1 = \frac{1}{2}.
\end{aligned}$$

If we choose $\alpha_2 = \gamma$, an arbitrary parameter, these equations can be solved exactly to give

$$\begin{aligned}
\alpha_2 &= \gamma, \\
\alpha_1 &= 1 - \gamma, \\
\beta_1 &= \beta_2 = \frac{1}{2\gamma}, \qquad \gamma \neq 0.
\end{aligned}$$

Combining all of this gives a one-step method of order $p = 2$ if $\gamma \neq 0$ and f is sufficiently smooth. We state this in the following algorithm.

Algorithm 2.2 *Runge-Kutta algorithm of order 2.*

To obtain an approximate solution of order $p = 2$ to the IVP (17), let $h = (b - a)/n$ and generate the sequences

$$\begin{aligned} y_{i+1} &= y_i + h\left[(1-\gamma)f(t_i, y_i) + \gamma f(t_i + \frac{h}{2\gamma}, y_i + \frac{h}{2\gamma} f(t_i, y_i))\right], \\ t_{i+1} &= t_i + h, \qquad i = 0, 1, \ldots, n-1, \end{aligned} \tag{34}$$

where $\gamma \neq 0$, $t_0 = a$, $y_0 = A$.

Euler's method is the special case, $\gamma = 0$, and has order 1; the improved Euler method has $\gamma = 1/2$ and the Euler-Cauchy method has $\gamma = 1$.

Example 9 *An illustration of the Euler-Cauchy method.*

Approximate the solution to $\dot{y} = y^2 + 1$, $y(0) = 0$ at $t = 0.1, 0.2, \ldots, 1.0$ using the Euler-Cauchy method with $h = 0.1$. The recurrence relation for y_{i+1} is

$$\begin{aligned} y_{i+1} &= y_i + hf(t_i + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, y_i)) \\ &= y_i + h\left[1 + (y_i + \frac{h}{2}(1 + y_i^2))^2\right], \end{aligned}$$

and the resulting approximations are given in Table 4. The IVP has the solution $y(t) = \tan t$. The approximate and exact solutions are represented graphically in Figure 3 where the approximating values y_i, $i = 0, 1, \ldots, 10$ have been joined by straight line segments.

If we increase n and tabulate the solution at $t = 1.0$, we obtain the results in Table 5, and we see that the error, $|y_n - \tan(1.0)| \approx 1.7h^2$ as h approaches zero.

A basic assumption in the derivation of the family of Runge-Kutta formulas (34) was that the solution $y(t)$ had three continuous derivatives. What if a formula of order 2 is used to solve an initial value problem whose solution has only two continuous derivatives, but not three. Examination of the local truncation error shows that the formula is then of order 1 and convergence is $O(h)$ and not $O(h^2)$. The point here is that higher order procedures can be used on problems whose solutions are not sufficiently smooth, but their rate of convergence may be reduced.

t_i	y_i	$\lvert y_i - \tan(t_i)\rvert$
0.0	0.00000	0.00e+00
0.1	0.10025	0.85e-04
0.2	0.20252	0.19e-03
0.3	0.30900	0.33e-03
0.4	0.42224	0.56e-03
0.5	0.54539	0.92e-03
0.6	0.68263	0.15e-03
0.7	0.83977	0.25e-02
0.8	1.02534	0.43e-02
0.9	1.25256	0.76e-02
1.0	1.54327	0.14e-01

Table 4: Approximate solution to $\dot{y} = y^2 + 1$, $y(0) = 0$, using the Euler-Cauchy method.

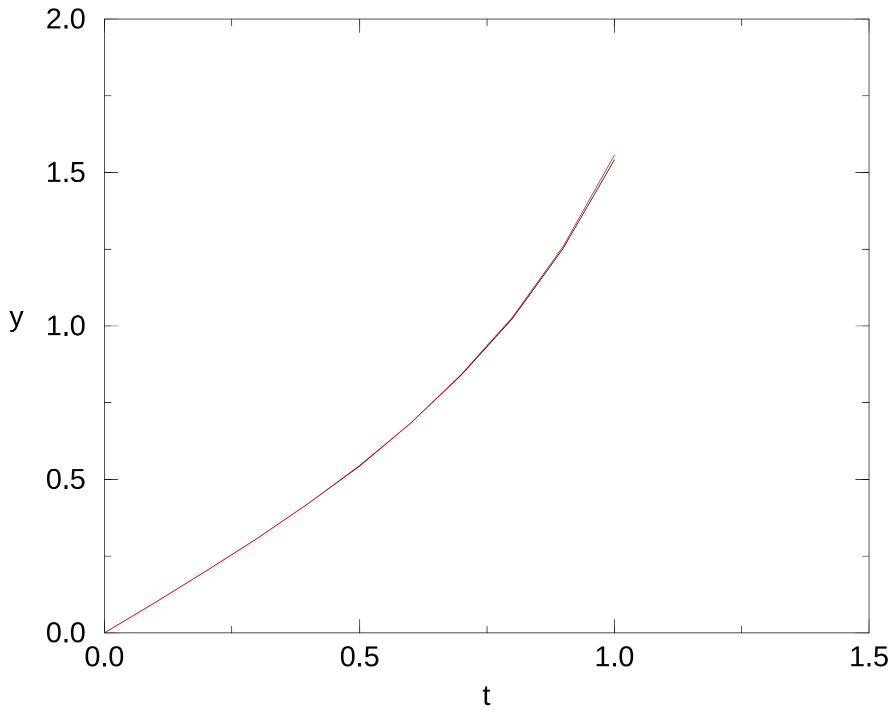

Figure 3: Analytical solution $y(t) = \tan t$ vs. numerical solution given at $t = 0.0, 0.1, \ldots, 1.0$, as in Table 5.

n	$h = 1/n$	y_n	$\|y_n - \tan(1)\|$	$\|y_n - \tan(1)\|/h$
10	0.10e+00	1.54327	0.14e-01	0.14e+01
100	0.10e-01	1.55724	0.17e-03	0.17e+01
1000	0.10e-02	1.55741	0.17e-03	0.17e+01

Table 5: Approximate solutions to $\dot{y} = y^2 + 1$, $y(0) = 0$ at $t = 1$ for small h using the Euler-Cauchy method.

Example 10 *An IVP whose solution is not smooth.*

A common example of problems whose solutions will not be smooth are those where the coefficients have a jump discontinuity at some point in the range of integration. In solving such problems numerically, integration should not be performed across the discontinuity. For example, suppose we are solving the problem

$$\ddot{y} + y = f(t), \quad 0 \le t \le 2,$$
$$y(0) = 0, \quad \dot{y}(0) = 1,$$
$$f(t) = \begin{cases} 1, \text{ for } 0 \le t \le 1, \\ 0, \text{ for } 1 \le t \le 2. \end{cases}$$

A good procedure would be to integrate from $t = 0$ to $t = 1$ and then from $t = 1$ to $t = 2$. On each subinterval, the differential equation has smooth solutions and convergence rates will be as advertised.

A major limitation of Runge-Kutta formulas is the amount of work required; work is measured in terms of the number of times the function f is evaluated. For higher order formulas, the work goes up dramatically; p evaluations per step lead to procedures of order p for $p = 1, 2, 3$, and 4, but not for 5; 6 evaluations are required for a formula of order 5, 7 for order 6, 9 for order 7, 11 for order 8, etc. For this reason, fourth order procedures are quite common. As in the second order case where the parameter γ was arbitrary, there is a family of fourth order formulas that depend on several parameters. One choice leads to the so-called *classical* formulas.

Algorithm 2.3 *Classical Runge-Kutta Formulas*

To obtain an approximate solution of order $p = 4$ to the IVP (17) on $[a, b]$, let $h = (b-a)/n$ and generate the sequences

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ t_{i+1} &= t_i + h, \qquad i = 0, 1, \ldots, n-1, \end{aligned} \tag{35}$$

where

$$\begin{aligned}
k_1 &= f(t_i, y_i),\\
k_2 &= f(t_i + h/2, y_i + (h/2)k_1),\\
k_3 &= f(t_i + h/2, y_i + (h/2)k_2),\\
k_4 &= f(t_i + h, y_i + hk_3),
\end{aligned}$$

and $t_0 = a$, $y_0 = A$.

As we pointed out earlier, the methods developed extend readily to systems of first order IVPs of the form (10). As an illustration, the formulas (35) in the case of an n-dimensional system become

$$\begin{aligned}
\mathbf{Y}_{i+1} &= \mathbf{Y}_i + \frac{h}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4),\\
t_{i+1} &= t_i + h, \qquad i = 0, 1, \ldots, n-1,
\end{aligned} \tag{36}$$

where

$$\begin{aligned}
\mathbf{K}_1 &= \mathbf{F}(t_i, \mathbf{Y}_i),\\
\mathbf{K}_2 &= \mathbf{F}(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{h}{2}\mathbf{K}_1),\\
\mathbf{K}_3 &= \mathbf{F}(t_i + \frac{h}{2}, \mathbf{Y}_i + \frac{h}{2}\mathbf{K}_2),\\
\mathbf{K}_4 &= \mathbf{F}(t_i + h, \mathbf{Y}_\mathbf{i} + h\mathbf{K}_3).
\end{aligned}$$

Example 11 *Classical Runge-Kutta method applied to a system of IVPs.*

Consider the system of first order IVPs,

$$\begin{aligned} \dot{Y}_1 &= Y_2, \\ \dot{Y}_2 &= Y_1 + t, \\ Y_1(0) &= 1, \quad Y_2(0) \ = \ 1, \end{aligned}$$

where, in terms of our previous notation,

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} Y_2 \\ Y_1 + t \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

So,

$$\mathbf{K}_1 = \mathbf{F}(t, \mathbf{Y}) = \begin{bmatrix} Y_2 \\ Y_1 + t \end{bmatrix},$$

which implies that

$$\mathbf{Y} + (h/2)\mathbf{K}_1 = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + (h/2) \begin{bmatrix} Y_2 \\ Y_1 + t \end{bmatrix} = \begin{bmatrix} Y_1 + (h/2)Y_2 \\ Y_2 + (h/2)(Y_1 + t) \end{bmatrix},$$

and therefore

$$\mathbf{K}_2 = \mathbf{F}(t + (h/2), \mathbf{Y} + (h/2)\mathbf{K}_1) = \begin{bmatrix} Y_2 + (h/2)(Y_1 + t) \\ Y_1 + (h/2)Y_2 + t + (h/2) \end{bmatrix},$$

etc. At the $(i+1)$st step, the values of t, Y_1, and Y_2 are assumed to be evaluated at the $(t_i, \mathbf{Y}_i)$.

See exercise set 3.

2.2.3 Some Implementation Issues

A too small value of h means that we are doing unnecessary computation that could lead to roundoff error (error induced because the arithmetic is not exact); a too large value of h means that we are probably not meeting the desired accuracy requirements (errors induced by discretization). Although we will not develop the details here, something can be said about the qualitative behavior of global error as a function of h for a one-step method of

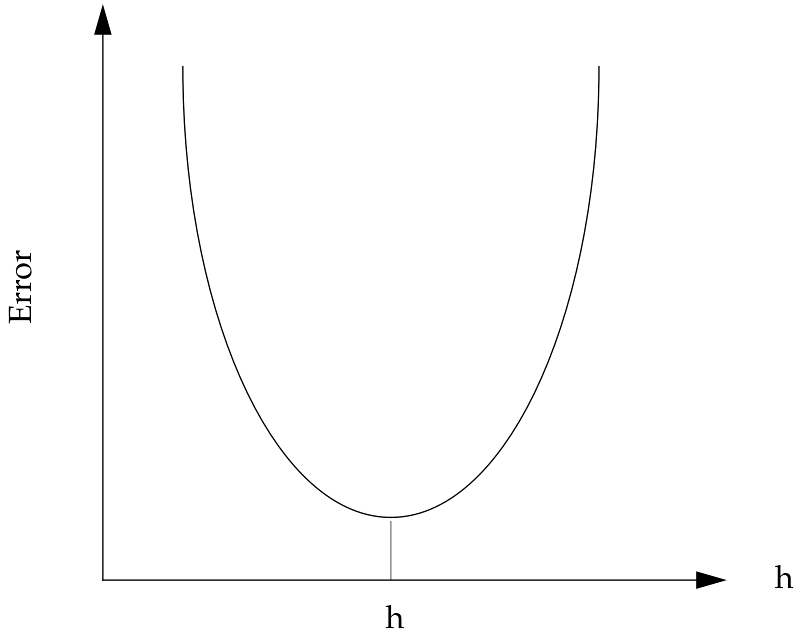

Figure 4: Qualitative behavior of absolute error at a fixed point as a function of the step size h.

order p. If the error in evaluating $\Delta(t_i, y_i)$ is ε_i with $|\varepsilon_i| \leq \varepsilon$ for all i and the error in forming $h\Delta(t_i, y_i)$ is ρ_i with $|\rho_i| \leq \rho$ for all i, then it can be shown that

$$|y(t_i) - y_i| \leq |y(t_0) - y_0|e^{L(b-a)} + \frac{e^{L(b-a)}}{L}\left(\frac{Ch^p}{2} + \varepsilon + \frac{\rho}{h}\right), \tag{37}$$

where C and L are constants depending on y and its derivatives. A graph of (37) has the qualitative behavior shown in Figure 3.

As we said earlier, most codes estimate local error at each step and attempt to adjust h accordingly. A widely used procedure for estimating local error follows. Suppose we have computed approximations of order p and $p+1$ at t_{i+1}, y_{i+1}^p and y_{i+1}^{p+1} respectively. Then using (20), the local error in the pth order approximation can be written as

$$d_{i+1} \;=\; u(t_{i+1}) - y_{i+1}^p \;=\; [u(t_{i+1}) - y_{i+1}^{p+1}] + [y_{i+1}^{p+1} - y_{i+1}^p]. \tag{38}$$

If h is sufficiently small, the term $[u(t_{i+1}) - y_{i+1}^{p+1}]$ can be neglected and we can use the computed value $[y_{i+1}^{p+1} - y_{i+1}^p]$ as an estimate of the error in the pth order formula. This sort of approximation has been validated by extensive numerical experimentation over the years.

2.2.4 The Codes RKSUITE

RKSUITE is an excellent collection of codes based on Runge-Kutta methods for the numerical solution of an IVP for a first order system of ODEs and is available in the public domain. It supersedes some very widely used codes written by the authors and their coauthors, namely the RKF45 code and its descendant DDERKF in the SLATEC library and DO2PAF and associated codes in the NAG Fortran library. RKSUITE is written in standard FORTRAN 77 and is distributed in source form. The advanced algorithms provide more functionality than is found in earlier codes, including new more efficient formulas, interpolation, automatic selection of the initial step size, a stiffness diagnostic, and global error assessment. The advanced software design includes a novel interface permitting both interval- and step-oriented solutions and is highly portable. The documentation includes detailed instructions for the effective use of the codes, a collection of templates illustrating the use of the codes for common tasks, output from running the templates in a number of variations, and comprehensive instructions for installing the codes.

RKSUITE implements three Runge-Kutta pairs: (2,3), (4,5), and (7,8). The (4,5) pair, for example, uses both a 4th and a 5th order approximation to estimate the error in the 4th order formula; using extrapolation, it then produces a formula of order 5. Similarly, the (2,3) pair produces a formula of order 3 and the (7,8) pair a formula of order 8.

The documentation for RKSUITE provided in RKSUITE.DOC is especially recommended. It is fairly short, carefully written document, explaining some of the internal workings of the suit of codes. The reader should pay particular attention to the description of *error control* and the effect on delivered accuracy. The documentation also contains an excellent set of references that describe the formulas and algorithms used in RKSUITE; references are also included that describe the design, implementation and testing of codes based on explicit Runge-Kutta formulas. The current code was written by R. Brankin (NAG). I. Gladwell (SMU) and L. Shampine (SMU) [7].

A copy of RKSUITE, together with sample drivers, can be obtained from "netlib" using anonymous ftp as follows:

- **ftp netlib.att.com** (login as anonymous and use your e-mail address as password)
- **bin** (set binary transfer mode)
- **cd netlib/ode/rksuite** (change to directory containing VODE codes)
- **mget *** (gets the compressed vode file)
- **quit** (exits ftp)
- **uncompress *.Z** (uncompress file *)

The file "templates" contains sample drivers for the codes in rksuite.

For a complete listing of what is available from "netlib", **mail netlib @ornl.gov** and type **send index** in the body of your message. The Internet address "netlib@ornl.gov" refers to a gateway machine at Oak Ridge National Laboratory in Oak Ridge, Tennessee. This address should be understood on all major networks.

2.2.5 RKSUITE Example

An important equation of nonlinear mechanics is van der Pol's equation,

$$\ddot{y} + \varepsilon(y^2 - 1)\dot{y} + y = 0, \tag{39}$$

for $\varepsilon > 0$. For any initial conditions, the solution of this equation converges to a unique periodic solution, called a stable limit cycle. The code below used RKSUITE to solve this problem with $\varepsilon = 1$, $y(0) = 1$, $\dot{y}(0) = 0$ for $0 \leq t \leq 12$. The code is followed by a tabulation of the solution at $t = 1, 2, \ldots, 12$ and a plot of the solution in the phase plane: $\dot{y}$ vs. y.

To put this problem in a form suitable for RKSUITE, we let $Y_1 = y$, $Y_2 = \dot{y}$ to obtain the equivalent system:

$$\begin{aligned}
\dot{Y}_1 &= Y_2, \\
\dot{Y}_2 &= -\varepsilon(Y_1^2 - 1)Y_2 - Y_1, \\
Y_1(0) = 1,\ Y_2(0) &= 0.
\end{aligned}$$

The system of ODEs is implemented in the subroutine F. The code uses the (4,5) Runge-Kutta pair.

```
C  Example to illustrate RKSUITE: the van der Pol equation
C
C     .. Parameters ..
      INTEGER           NEQ, LENWRK, METHOD
      PARAMETER         (NEQ=2,LENWRK=32*NEQ,METHOD=2)
C     .. Local Scalars ..
      DOUBLE PRECISION  HNEXT, HSTART, T, TEND, TINC, TLAST,
     &                  TOL, TSTART, TWANT, WASTE
      INTEGER           L, NOUT, STPCST, STPSOK, TOTF, UFLAG
      LOGICAL           ERRASS, MESAGE
C     .. Local Arrays ..
      DOUBLE PRECISION  THRES(NEQ), WORK(LENWRK), Y(NEQ), YMAX(NEQ),
     &                  YP(NEQ), YSTART(NEQ)
C     .. External Subroutines ..
      EXTERNAL          F, SETUP, STAT, UT
C     .. Executable Statements ..
C
C  Set the initial conditions.  Take TEND well past the last
C  output point, TLAST.  When this is possible, and it usually
C  is, it's good practice.
C
      TSTART = 0.0D0
```

```
      YSTART(1) = 1.0D0
      YSTART(2) = 0.0D0
      TLAST = 12.0D0
      TEND = TLAST +1.0D0
C
C  Initialize output.
C
      WRITE (*,'(A,I10)')  'Solution with METHOD = ', METHOD
      WRITE (*,'(/A/)')    '    t           y       dy/dt'
      WRITE (*,'(1X,F6.3,3X,F9.4,3X,F9.4)')  TSTART, YSTART(1),
     &  YSTART(2)
C
C  Set error control parameters.
C
      TOL = 5.0D-5
      DO 20 L = 1, NEQ
         THRES(L) = 1.0D-5
   20 CONTINUE
C
C  Call the setup routine. Because messages are requested, MESAGE = .TRUE.,
C  there is no need later to test values of flags and print out explanations.
C  In this variant no error assessment is done, so ERRASS is set .FALSE..
C  By setting HSTART to zero, the code is told to find a starting (initial)
C  step size automatically .
C
      MESAGE = .TRUE.
      ERRASS = .FALSE.
      HSTART = 0.0D0
      CALL SETUP(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,'Usual Task',
     &           ERRASS,HSTART,WORK,LENWRK,MESAGE)
C
C  Compute answers at NOUT equally spaced output points. It is good
C  practice to code the calculation of TWANT so that the last value
C  is exactly TLAST.
C
      NOUT = 12
      TINC = (TLAST-TSTART)/NOUT
C
      DO 40 L = 1, NOUT
         TWANT = TLAST + (L-NOUT)*TINC
         CALL UT(F,TWANT,T,Y,YP,YMAX,WORK,UFLAG)
C
```

```
         IF (UFLAG.GT.2) GO TO 60
C
C  Success. T = TWANT. Output computed and true solution components.
C
         WRITE (*,'(1X,F6.3,3X,F9.4,3X,F9.4)') T, Y(1), Y(2)
   40 CONTINUE
C
C  The integration is complete or has failed in a way reported in a
C  message to the standard output channel.
C
   60 CONTINUE
C
C  YMAX(L) is the largest magnitude computed for the solution component
C  Y(L) in the course of the integration from TSTART to the last T.  It
C  is used to decide whether THRES(L) is reasonable and to select a new
C  THRES(L) if it is not.
C
      WRITE (*,'(A/)') '               YMAX(L) '
      DO 80 L = 1, NEQ
         WRITE (*,'(13X,1PE8.2)')    YMAX(L)
   80 CONTINUE
C
C  The subroutine STAT is used to obtain some information about the progress
C  of the integration. TOTF is the total number of calls to F made so far
C  in the integration; it is a machine-independent measure of work.  At present
C  the integration is finished, so the value printed out refers to the overall
C  cost of the integration.
C
      CALL STAT(TOTF,STPCST,WASTE,STPSOK,HNEXT)
      WRITE (*,'(/A,I10)')
     &  ' The cost of the integration in evaluations of F is', TOTF
C
      STOP
      END
C
      SUBROUTINE F(T,Y,YP)
C     .. Scalar Arguments ..
      DOUBLE PRECISION  T, EPS
      PARAMETER (EPS=1.0D0)
C     .. Array Arguments ..
      DOUBLE PRECISION  Y(*), YP(*)
C     .. Executable Statements ..
```

```
      YP(1) =  Y(2)
      YP(2) = - EPS * (Y(1)**2 - 1.0D0) * Y(2) -Y(1)
      RETURN
      END
```

```
    t          y       dy/dt

   .000     1.0000      .0000
  1.000      .4976    -1.0442
  2.000    -1.1962    -1.8675
  3.000    -1.7280      .4147
  4.000     -.9569     1.1587
  5.000      .9870     2.6183
  6.000     1.9549     -.3356
  7.000     1.3093     -.9156
  8.000     -.1653    -2.3296
  9.000    -2.0004     -.1947
 10.000    -1.5820      .7342
 11.000     -.4910     1.6547
 12.000     1.7471     1.4651
            YMAX(L)

            2.00E+00
            2.67E+00
```

See exercise set 4.

2.3 Multi-Step Methods

The Taylor Series and explicit Runge-Kutta methods that we have discussed so far have no memory: the value of $y(t)$ for t before t_i do not directly affect the values of $y(t)$ for t after t_i. Other methods take advantage of previously computed solution values and are referred to as multistep methods. The Adam's formulas for non-stiff problems and the Backward Differentiation Formulas for stiff problems furnish important and widely-used examples of multi-step methods.

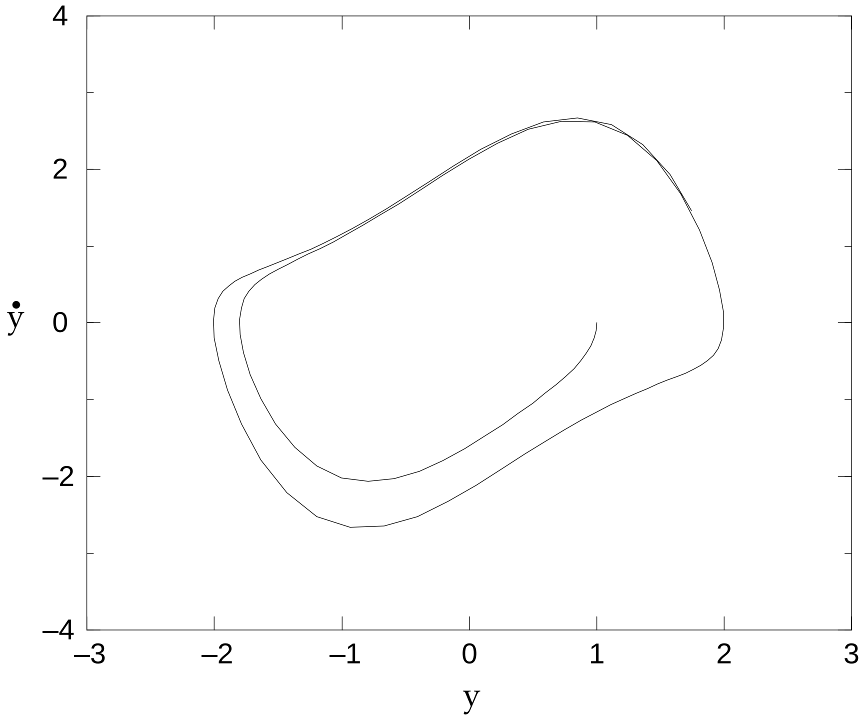

Figure 5: The van der Pol equation: $\ddot{y} + (y^2 - 1)\dot{y} + y = 0$, where $y(0) = 1$ and $\dot{y}(0) = 0$.

2.3.1 The Adams-Bashforth and Adams-Moulton Formulas

On reaching a mesh point t_i with approximate solution $y_i \cong y(t_i)$, there are (usually) available approximate solutions $y_{i+1-j} \cong y(t_{i+1-j})$ for $j = 2, 3, \ldots, p$. From the differential equation itself, approximations to the derivatives $\dot{y}(t_{i+1-j})$ can be obtained from

$$\dot{y}_{i+1-j} = f(t_{i+1-j}, y(t_{i+1-j})) \cong f(t_{i+1-j}, y_{i+1-j}) = f_{i+1-j}. \tag{40}$$

This information can be exploited for solution values prior to the current point t_i by using the integrated form of the differential equation:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} \dot{y}(t)dt = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))dt. \tag{41}$$

The Adams Bashforth formula of order p is obtained by integrating the polynomial $P(t)$ that interpolates f_{i+1-j} at t_{i+1-j} for $j = 1, 2, \ldots, p$, in place of f:

$$y_{i+1} = y_i + h \sum_{j=1}^{p} \alpha_{p,j} f_{i+1-j}. \tag{42}$$

Formula (42) involves only one evaluation of f per step. An attractive feature of the approach is the underlying polynomial approximation $P(t)$ to $\dot{y}(t)$ because it can be used to approximate $y(t)$ between mesh points:

$$y(t) \cong y_i + \int_{t_i}^{t} P(t)dt. \tag{43}$$

The lowest order Adams-Bashforth formula arises from interpolating the single value $f_i = f(t_i, y_i)$ by $P(t)$. The interpolating polynomial is constant so its integration from t_i to t_{i+1} results in $hf(t_i, y_i)$ and the first order Adams-Bashforth formula (AB1):

$$y_{i+1} = y_i + hf(t_i, y_i). \tag{44}$$

This is just the familiar forward Euler formula. For constant step size h, the second order Adams-Bashforth formula (AB2) is also easily found to be

$$y_{i+1} = y_i + h[(3/2)f(t_i, y_i) - (1/2)f(t_{i-1}, y_{i-1})]. \tag{45}$$

The implicit Adams-Moulton formulas arises when the polynomial $P(t)$ interpolates f_{i+1-j} for $j = 0, 1, \ldots, p-1$:

$$y_{i+1} \quad = \quad y_i + h \sum_{j=1}^{p-1} \hat{\alpha}_{p,j} f_{i+1-j}. \tag{46}$$

When $j = p-1$, the right hand side contains the term $f_{i+1} = f(t_{i+1}, y_{i+1})$, and we see that y_{i+1} is defined only implicitly by this formula. The solution is accomplished by first "predicting" the result using the explicit Adams-Bashforth formula (42), and then "correcting" it using the implicit formula (46); we then proceed by "simple" or "functional" iteration. If L is a bound on $|\partial f/\partial y|$ and the step size h is small enough so that for some constant ρ,

$$|h\hat{\alpha}_{k,0}|L \le \rho < 1,$$

then (46) has a unique solution y_{i+1} and the error is decreased by a factor of ρ at each iteration. For "small" step sizes h, the iteration converges very quickly.

Note that for a formula of order p, both the Adams-Bashforth and Adams-Moulton formulas interpolate the function f on p t-points. The t-points overlap and are illustrated graphically below in Figure 7.

The lowest order Adams-Moulton formula involves interpolating the single value $f_{i+1} = f(x_{i+1}, y_{i+1})$ and an easy calculation leads to the formula

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}), \tag{47}$$

which defines y_{i+1} implicitly. The resulting formula is called the backward Euler formula. From its definition it is clear that it has the same accuracy as the forward Euler method; its advantage is vastly superior stability. The second order Adams-Moulton method also

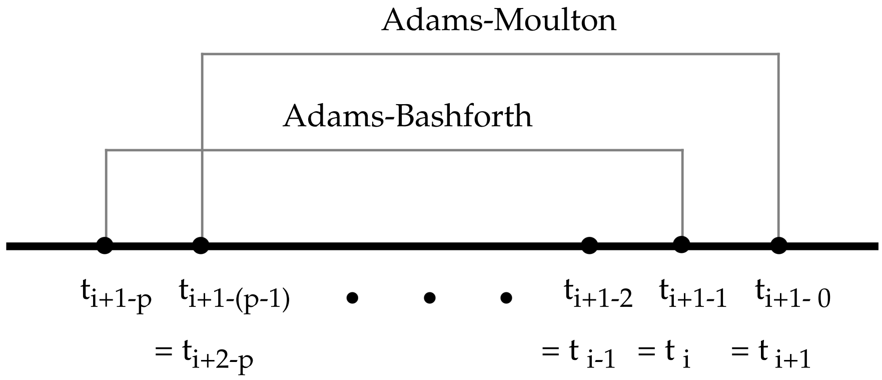

Figure 6: t-points at which order p Adams-Bashforth and Adams-Moulton formulas interpolate f.

does not use previously computed solution values; it is called the trapezoidal rule because it generalizes the trapezoidal rule for integrals to differential equations:

$$y_{i+1} = y_i + \frac{h}{2}\left[f(t_{i+1}, y_{i+1}) + f(t_i, y_i)\right]. \tag{48}$$

The third order formula is more typical because it does involve a previously computed value. When the step size is a constant h, it is

$$y_{i+1} = y_i + h\left[(5/12)f(t_{i+1}, y_{i+1}) + (8/12)f(t_i, y_i) - (1/12)f(t_{i-1}, y_{i-1})\right]. \tag{49}$$

The Adams-Moulton formula of order p is more accurate than the Adams-Bashforth formula of the same order, so that it can use a larger step size; the Adams-Moulton formula is also more stable. A modern code based on such methods is more complex than a Runge-Kutta code because it must cope with the difficulties of starting the integration and changing the step size. With enough "memorized" values, however, we can use whatever order formula we wish in the step from t_i. Modern Adams codes attempt to select the most efficient formula at each step as well as to choose an optimal step size h to achieve a user-specified accuracy.

Some general rules-of-thumb about how to choose between Runge-Kutta methods and Adams methods for solving nonstiff problems are given below:

- Generally, Adams methods are superior if output at many points is needed.
- If function evaluations are **expensive**, Adams methods are preferred.
- If function evaluations are **inexpensive** and moderate accuracy is required, Runge-Kutta methods are generally best.
- If storage is at a premium, Runge-Kutta methods win.
- If accuracy over a wide range of tolerances is needed, the variable order Adams methods will outperform the fixed order Runge-Kutta methods.

Recent developments in Runge-Kutta methods have shifted these boundaries somewhat; RKSUITE, for example, has an interpolation capability that makes it more efficient than the previous generation of Runge-Kutta codes and the (7,8) solution pair is very efficient at stringent error tolerances.

An excellent discussion of Adams methods as well as a widely used suite of codes is given by Shampine and Gordon in [4]. Gear's test [5] presents a variety of methods, and is a primary source about the solution of stiff problems to be discussed in the next section.

2.3.2 Stiff Problems: Backward Differentiation Formulas

A problem is stiff if the numerical solution has its step size limited more severely by the stability of the numerical technique than by the accuracy of the technique. Frequently, these problems occur in systems of differential equations that involve several components that are decaying at widely differing rates. The reader is encouraged to consult the excellent survey article [6] by Shampine and Gear, "A User's View of Solving Stiff Ordinary Differential Equations."

A simple scalar example of stiffness is given by C.W. Gear [5]:

$$\dot{y} = \lambda\,[y - F(t)] + \dot{F}(t), \qquad \lambda \ll 0, \tag{50}$$

where $F(t)$ is a smooth, slowly varying function. The solution,

$$y = [y_0 - F(0)]\, e^{\lambda t} + F(t), \tag{51}$$

has a component $[y_0 - F(0)]\, e^{\lambda t}$ that will be insignificant compared to $F(t)$ for t sufficiently large. The numerical method, however, will always have its step size h limited by the magnitude of λh for the entire integration.

Example 12 *Gear's example.*

In Gear's example above, choose $F(t) = t$. In Table 7 we tabulate the cost of integrating the IVP,

$$\dot{y} \;=\; \lambda(y - t) + 1, \qquad 0 \le t \le 10,\; y(0) = 1,$$

using RKSUITE for $\lambda = -10, -20, -30, -100$. The solution is $y(t) = e^{\lambda t} + t$. Relative and absolute error tolerances were taken to be 10^{-6}.

Note that the smaller the solution component $e^{\lambda t}$ becomes, the harder RKSUITE works. Using a code designed specifically for stiff problems such as the code VODE presented in the next section, the number of function evaluations would have been approximately 120 for the range of λ-values considered.

λ	Number of function evaluations
-10	339
-20	583
-30	849
-100	2528

Table 6: Solution of a stiff IVP using RKSUITE.

A class of multi-step formulas which are highly effective in solving stiff problems are based on numerical differentiation. Again, we start by interpolating the previously computed solution values $y_i, y_{i-1}, \ldots, y_{i-p}$ as well as the new one y_{i+1} by a polynomial $P(t)$. The derivative of the solution at t_{i+1} is then approximated by $\dot{P}(t_{i+1})$. The approximation is related to the differential equation by insisting that it satisfy the differential equation at t_{i+1}:

$$\dot{P}(t_{i+1}) \quad = \quad f(t_{i+1}, P(t_{i+1})) \quad = \quad f(t_{i+1}, y_{i+1}).$$

Substituting for $\dot{P}(t_{i+1})$ in this equation, we obtain the family of backward differentiation formulas, the BDFs:

$$\bar{\alpha}_0 y_{i+1} + \bar{\alpha}_1 y_i + \ldots + \bar{\alpha}_p y_{i+1-p} \quad = \quad hf(t_{i+1}, y_{i+1}). \tag{52}$$

These formulas were popularized by Gear, and are sometimes known as Gear's formulas. They are implicit like the Adams-Moulton formulas, but not as accurate for formulas of the same order and not stable for orders 7 and up. However, at the orders for which the formulas are stable, they are **much** more stable than the Adams-Moulton formulas. The formulas (52) cannot be evaluated by simple iteration because this restricts the step size just as much as stability does for much less stable formulas. In practice, a modified Newton iteration (Linear Algebra Chapter) is used to solve the nonlinear algebraic equations for y_{i+1}; this requires approximating partial derivatives and solving systems of linear equations.

As with Adams formulas, modern codes based on the BDFs vary the formula (the order used) as well as the step size. The solution of problems that are quite stiff are completely impractical with a method intended for non-stiff problems, such as an explicit Runge-Kutta method or an Adams-Moulton method evaluated by simple iteration

The simplest BDF is when $P(t)$ is the straight line interpolating y_{i+1} and y_i. The derivative at t_{i+1} is the constant slope of this line and setting it to $f(t_{i+1}, y_{i+1})$ results in

$$\frac{y_{i+1} - y_i}{h} = f(t_{i+1}, y_{i+1}). \tag{53}$$

Once again we have derived the backward Euler formula! Although this case results in a one-step formula, the higher order BDFs do involve previously computed solution values.

For example, when the step size is a constant h, the backward differentiation formula of order two is

$$y_{i+1} = (4/3)y_i - (1/3)y_{i-1} + h(2/3)f(t_{i+1}, y_{i+1}). \tag{54}$$

A code providing a highly efficient implementation of the Adam's formulas and the BDF formulas is given in the next section.

2.3.3 The Code VODE

VODE is a relatively new initial value ODE solver for stiff and non stiff problems and was written by P.N. Brown (LLNL), G.D. Byrne (currently at SMU), and A.C. Hindmarsh (LLNL). It uses variable coefficient Adams-Moulton and Backward Differentiation Formula (BDF) methods. The initial step size is selected internally and a method of order 1 is used to start the integration; the order is increased as sufficient data become available. The setting, MF $= 10$, is used to request a solution using the Adams method and setting of MF $= 21, 22, 24$, or 25 requests the BDF method. A copy of the code may be obtained from "netlib" as follows:

- **ftp netlib.att.com** (login as anonymous and use your e-mail address as password)
- **bin** (set binary transfer mode)
- **cd netlib/ode** (change to directory containing VODE codes)
- **get vode.f.Z** (gets the compressed vode file)
- **quit** (exits ftp)
- **uncompress *.Z** (uncompress file *)

The reader should consult the SIAM article [8] for a discussion of VODE. The article also contains a couple of interesting test cases and a good set of references for further reading. The code itself is well documented; leading comments contain a sample driver and details about the code input and output parameters.

2.3.4 VODE Example

The following IVP due to Robertson (1966) arises in the study of chemical kinetics:

$$\begin{aligned}
\dot{Y}_1 &= -0.04Y_1 + 10^4 Y_2 Y_3, \\
\dot{Y}_2 &= 0.04Y_1 - 10^4 Y_2 Y_3 - 3 \times 10^7 Y_2^2, \\
\dot{Y}_3 &= 3 \times 10^7 Y_2^2, \\
Y_1(0) &= 1,\ Y_2(0) = 0,\ Y_3(0) = 0.
\end{aligned}$$

The problem is stiff and the code below uses VODE to solve it on the interval, $0 \leq t \leq 4 \times 10^{10}$. For an explanation of the various input parameters, see the prologue to VODE. Since the solution of a linear system is required in the BDF method, the Jacobian of **F** must be available. VODE allows the Jacobian to be supplied by the user (MF = 21 or 24) or to be generated internally by the code (MF = 22 or 25). It is always preferable to supply the Jacobian if possible. Since

$$\mathbf{F} = \begin{bmatrix} -0.04Y_1 + 10^4 Y_2 Y_3 \\ 0.04Y_1 - 10^4 Y_2 Y_3 - 3 \times 10^7 Y_2^2 \\ 3 \times 10^7 Y_2^2 \end{bmatrix},$$

the Jacobian is

$$\mathbf{J} \;=\; (\partial F_i/\partial Y_j) \;=\; \begin{bmatrix} -0.04 & 10^4 Y_3 & 10^4 Y_2 \\ 0.04 & -10^4 Y_3 - 6 \times 10^7 Y_2 & -10^4 Y_2 \\ 0 & 6 \times 10^7 Y_2 & 0 \end{bmatrix}.$$

F is implemented in the subroutine FEX and **J** in the subroutine JEX. We note that although the Jacobian was not difficult to calculate here, it can be tedious in many applications; modern symbolic packages, such as MAPLE, MATHEMATICA, etc. make it easier to supply **J**.

The code is followed by two tabulations of the solution at $t = 4\times10^{-1}, 4\times10^{0}, 4 \times 10^{1}, \ldots, 4\times10^{10}$: the first for a supplied Jacobian (MF=21) and the second for an internally generated Jacobian (MF=22). Both methods produce acceptably accurate solutions, but more work is required for the internally-generated Jacobian case. The quantity SUM $= Y_1 + Y_2 + Y_3$ is also printed out. Since $\dot{Y}_1(t) + \dot{Y}_2(t) + \dot{Y}_3(t) = 0$, SUM should have the value "1." Note that this condition is necessary for accuracy, but not for sufficient. The output is followed by a combined plot of $Y_1(t)$, $Y_2(t)$, and $Y_3(t)$ vs. t for $0 \le t \le 4 \times 10^{10}$.

```
C Example to illustrate VODE: a problem from chemical kinetics
C
C The following code is set up to use VODE to solve the following
C problem:
C
C     dy1/dt = -.04*y1 + 1.e4*y2*y3
C     dy2/dt = .04*y1 - 1.e4*y2*y3 - 3.e7*y2**2
C     dy3/dt = 3.e7*y2**2
C
C on the interval from t = 0.0 to t = 4.e10, with initial conditions
C y1 = 1.0, y2 = y3 = 0.  The problem is stiff.
C
C The code uses MF = 21 (the Jacobian is supplied) and the results
C are printed at t = .4, 4., ..., 4.e10.  The values ITOL = 2 is
C used, with ATOL much smaller for y2 than y1 or y3 because
```

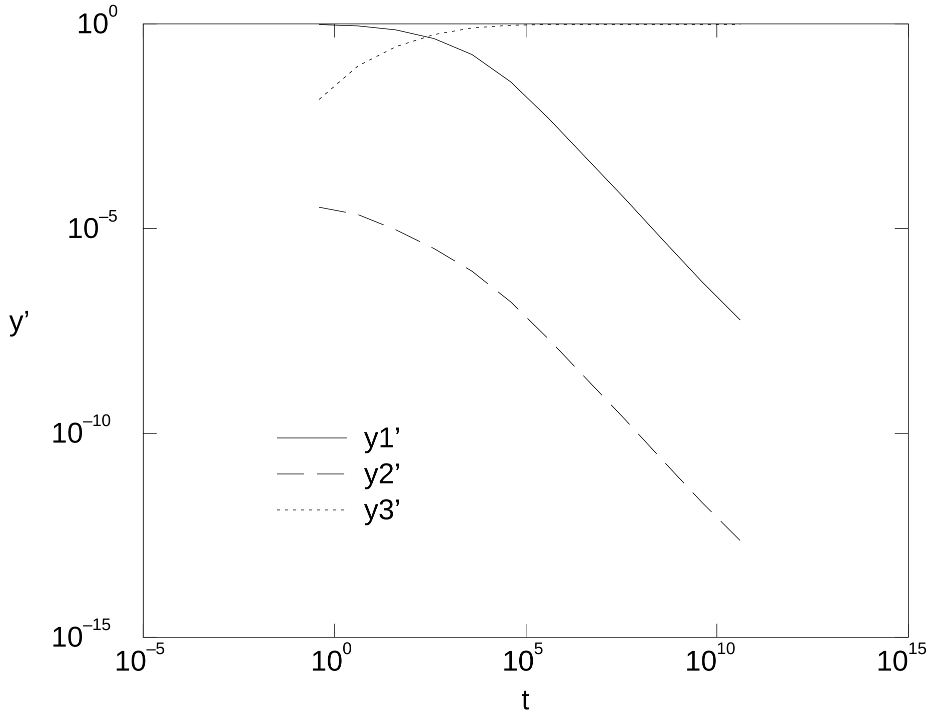

Figure 7: Output from the VODE solution to Robertson's chemical kinetics problem.

```
C y2 has much smaller values
C
C At the end of the run, statistical quantities of interest are
C printed.  For a complete description of all input and output
C quantities, see the comments in the beginning of the code VODE.
C
      EXTERNAL FEX, JEX
      DOUBLE PRECISION ATOL, RPAR, RTOL, RWORK, T, TOUT, Y
      DIMENSION Y(3), ATOL(3), RWORK(67), IWORK(33)
C
C Set initial conditions and control parameters.
C
      NEQ = 3
      Y(1) = 1.0D0
      Y(2) = 0.0D0
      Y(3) = 0.0D0
      T = 0.0D0
      TOUT = 0.4D0
C
      ITASK = 1
      ISTATE = 1
```

```
      IOPT = 0
      LRW = 67
      LIW = 33
      MF = 21
C
C Set error control parameters.
C
      ITOL = 2
      RTOL = 1.D-4
      ATOL(1) = 1.D-8
      ATOL(2) = 1.D-14
      ATOL(3) = 1.D-6
C
C Initialize output.
C
      SUM = Y(1) + Y(2) + Y(3)
      WRITE(6,20)T,Y(1),Y(2),Y(3),SUM
  20  FORMAT(' t =',1PD10.2,'  y =',3D14.6,'  sum =',1PD10.2)
C
C Solution.
C
      DO 40 IOUT = 1,12
         CALL DVODE(FEX,NEQ,Y,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
     1             IOPT,RWORK,LRW,IWORK,LIW,JEX,MF,RPAR,IPAR)
         SUM = Y(1) + Y(2) + Y(3)
         WRITE(6,20)T,Y(1),Y(2),Y(3),SUM
         IF (ISTATE .LT. 0) GO TO 80
  40     TOUT = TOUT*10.
      WRITE(6,60) IWORK(11),IWORK(12),IWORK(13),IWORK(19),
     1            IWORK(20),IWORK(21),IWORK(22)
  60  FORMAT(/' No. steps =',I4,'   No. f-s =',I4,
     1       '   No. J-s =',I4,'   No. LU-s =',I4/
     2       '  No. nonlinear iterations =',I4/
     3       '  No. nonlinear convergence failures =',I4/
     4       '  No. error test failures =',I4/)
      STOP
  80  WRITE(6,90)ISTATE
  90  FORMAT(///' Error halt.. ISTATE =',I3)
      STOP
      END
C
C
```

```
      SUBROUTINE FEX (NEQ, T, Y, YDOT, RPAR, IPAR)
      DOUBLE PRECISION RPAR, T, Y, YDOT
      DIMENSION Y(NEQ), YDOT(NEQ)
      YDOT(1) = -4.0D-2*Y(1) + 1.0D4*Y(2)*Y(3)
      YDOT(3) = 3.0D7*Y(2)*Y(2)
      YDOT(2) = -YDOT(1) - YDOT(3)
      RETURN
      END
C
      SUBROUTINE JEX (NEQ, T, Y, ML, MU, PD, NRPD, RPAR, IPAR)
      DOUBLE PRECISION PD, RPAR, T, Y
      DIMENSION Y(NEQ), PD(NRPD,NEQ)
      PD(1,1) = -4.0D-2
      PD(1,2) = 1.0D4*Y(3)
      PD(1,3) = 1.0D4*Y(2)
      PD(2,1) = 4.0D-2
      PD(2,3) = -PD(1,3)
      PD(3,2) = 6.0D7*Y(2)
      PD(2,2) = -PD(1,2) - PD(3,2)
      RETURN
      END
```

VODE output for user supplied Jacobian (MF = 21).

```
 t =  0.00D+00  y =  1.000000D+00  0.000000D+00  0.000000D+00  sum =  1.00D+00
 t =  4.00D-01  y =  9.851641D-01  3.386242D-05  1.480205D-02  sum =  1.00D+00
 t =  4.00D+00  y =  9.055097D-01  2.240338D-05  9.446793D-02  sum =  1.00D+00
 t =  4.00D+01  y =  7.157956D-01  9.183493D-06  2.841952D-01  sum =  1.00D+00
 t =  4.00D+02  y =  4.505029D-01  3.222678D-06  5.494939D-01  sum =  1.00D+00
 t =  4.00D+03  y =  1.831963D-01  8.942025D-07  8.168028D-01  sum =  1.00D+00
 t =  4.00D+04  y =  3.899263D-02  1.622200D-07  9.610072D-01  sum =  1.00D+00
 t =  4.00D+05  y =  4.936230D-03  1.984192D-08  9.950638D-01  sum =  1.00D+00
 t =  4.00D+06  y =  5.169361D-04  2.068797D-09  9.994831D-01  sum =  1.00D+00
 t =  4.00D+07  y =  5.202403D-05  2.081068D-10  9.999480D-01  sum =  1.00D+00
 t =  4.00D+08  y =  5.214998D-06  2.086010D-11  9.999948D-01  sum =  1.00D+00
 t =  4.00D+09  y =  5.204478D-07  2.081792D-12  9.999995D-01  sum =  1.00D+00
 t =  4.00D+10  y =  5.373726D-08  2.149490D-13  9.999999D-01  sum =  1.00D+00

 No. steps = 559   No. f-s = 852   No. J-s =  11   No. LU-s = 112
  No. nonlinear iterations = 816
  No. nonlinear convergence failures =   0
```

```
No. error test failures =  25
```

VODE output for internally generated Jacobian (MF=22).

```
t =  0.00D+00  y =  1.000000D+00  0.000000D+00  0.000000D+00  sum =  1.00D+00
t =  4.00D-01  y =  9.851641D-01  3.386242D-05  1.480205D-02  sum =  1.00D+00
t =  4.00D+00  y =  9.055097D-01  2.240338D-05  9.446793D-02  sum =  1.00D+00
t =  4.00D+01  y =  7.157944D-01  9.183473D-06  2.841964D-01  sum =  1.00D+00
t =  4.00D+02  y =  4.505138D-01  3.223026D-06  5.494830D-01  sum =  1.00D+00
t =  4.00D+03  y =  1.832254D-01  8.943769D-07  8.167737D-01  sum =  1.00D+00
t =  4.00D+04  y =  3.898040D-02  1.621635D-07  9.610194D-01  sum =  1.00D+00
t =  4.00D+05  y =  4.937593D-03  1.984717D-08  9.950624D-01  sum =  1.00D+00
t =  4.00D+06  y =  5.167596D-04  2.068094D-09  9.994832D-01  sum =  1.00D+00
t =  4.00D+07  y =  5.201559D-05  2.080730D-10  9.999480D-01  sum =  1.00D+00
t =  4.00D+08  y =  5.188591D-06  2.075447D-11  9.999948D-01  sum =  1.00D+00
t =  4.00D+09  y =  5.197009D-07  2.078805D-12  9.999995D-01  sum =  1.00D+00
t =  4.00D+10  y =  5.966719D-08  2.386688D-13  9.999999D-01  sum =  1.00D+00

No. steps = 573   No. f-s = 859   No. J-s =  12   No. LU-s = 127
 No. nonlinear iterations = 856
 No. nonlinear convergence failures =   0
 No. error test failures =  39
```

See exercise sets 5 and 6.

---

3 Exercises

Exercises to Illustrate the Theory of ODEs

— Exercise Set 1

1.1 Write down an equivalent first order system of differential equations for each of the following higher order equations:

$$
\begin{aligned}
&a)\quad \ddot{y} = f(t, y).\\
&b)\quad \ddot{x} + x = f(t, x).\\
&c)\quad \ddot{\theta} + (g/l)\sin\theta = F(t).\\
&d)\quad \ddot{x} + \varepsilon(x^2 - 1)\dot{x} + x = 0.\\
&e)\quad L\ddot{Q} + r\dot{Q} + Q/C = E(t).\\
&f)\quad x^{(4)} + e^t\dot{x} - tx = \cos t.\\
&g)\quad m\ddot{x} = X(t, x, y, z, \dot{x}, \dot{y}, \dot{z}),\\
&\qquad m\ddot{y} = Y(t, x, y, z, \dot{x}, \dot{y}, \dot{z}),\\
&\qquad m\ddot{z} = Z(t, x, y, z, \dot{x}, \dot{y}, \dot{z}).\\
&h)\quad u^{(6)} + u\dot{u} = e^t.
\end{aligned}
$$

1.2 Put each of the following IVPs in the form (10):

$$
\begin{aligned}
&a)\quad \ddot{y} + 4\dot{y} + 8y = \sin t,\ \ y(0) = 1, \dot{y}(0) = 0.\\
&b)\quad \ddot{y} + \dot{y} = \log t,\ \ y(1) = 0, \dot{y}(1) = -1.\\
&c)\quad z^{(3)} - \ddot{z}\sin t - 2\dot{z}\cos t + z\sin t = \log t,\\
&\qquad z(1) = A_1, \dot{z}(1) = A_2, \ddot{z}(1) = A_3.\\
&d)\quad \ddot{s} + 0.042\dot{s} + 0.961s = \dot{\theta} + 0.063\theta,\\
&\qquad \ddot{u} + 0.087\dot{u} = \dot{s} + 0.025s,\\
&\qquad \dot{v} = 0.873(u - v), \dot{w} = 0.433(v - w),\\
&\qquad \dot{x} = 0.058(w - x), \dot{\theta} = -0.396(x - 47.6),\\
&\qquad s(0) = \dot{s}(0) = \dot{u}(0) = \theta(0) = 0, u(0) = 50, v(0) = w(0) = x(0) = 75.
\end{aligned}
$$

1.3 Consider the following systems of equations:

$$
\begin{aligned}
&a)\quad \dot{Y}_1 = Y_2,\ \ \dot{Y}_2 = -t^2 Y_1 - tY_2.\\
&b)\quad \dot{Y}_1 = e^{-t^2/2} Y_2,\ \ \dot{Y}_2 = -t^2 e^{t^2/2} Y_1.\\
&c)\quad \dot{Y}_1 = -\frac{t}{2} Y_1 + Y_2,\ \ \dot{Y}_2 = \left(\frac{1}{2} - \frac{3t^2}{4}\right) Y_1 - \frac{t}{2} Y_2.
\end{aligned}
$$

Verify for each system that $Y_1(t) = y(t)$, where $y(t)$ satisfies the second order equation

$$\ddot{y} + t\dot{y}(t) + t^2 y(t) = 0. \tag{55}$$

1.4 In Example 5c, show that the quantity $|\partial f/\partial y|$ is not bounded as y approaches 1; hence, the corresponding solution may fail to exist or not be unique. What is the situation in 5b?

Exercises to Illustrate Taylor Series Methods

— Exercise Set 2

2.1 Use the Taylor series algorithm of orders $p = 1$ (Euler's method) and $p = 2$ with $h = 0.1$ to compute an approximate solution at $t = 0.5$ to each of the following IVPs. In each case find the exact solution and compare it with your numerical result.

$$\begin{aligned}
&a)\ \dot{y} = 2y - 1,\ y(0) = 1.\\
&b)\ \dot{y} = y - t,\ y(0) = 2.\\
&c)\ \dot{y} - y = \sin 2t,\ y(0) = 0.\\
&d)\ \dot{y} = y^2 - 3y + 2,\ y(0) = 2.\\
&e)\ \dot{y} = 1 + y^2,\ y(0) = 0.\\
&f)\ \dot{y} + ty^2 = 0,\ y(0) = 2.\\
&g)\ \dot{y} = t,\ y(0) = 0.\\
&h)\ \dot{y} = -y,\ y(0) = 1.\\
&i)\ \dot{y} = t/y,\ y(0) = 1.\\
&j)\ \dot{y} = y(1-y),\ y(0) = 2.
\end{aligned}$$

2.2 For the linear equation, $\dot{y} = P_1(t)y + Q_1(t)$, show that the derivatives needed in the Taylor series algorithm can be obtained from the recursion,

$$y^{(r)} = P_r(t)y + Q_r(t), \tag{56}$$

where

$$\begin{aligned}
P_r(t) &= \dot{P}_{r-1}(t) + P_1(t)P_{r-1}(t),\\
Q_r(t) &= \dot{Q}_{r-1}(t) + Q_1(t)P_{r-1}(t), \qquad r = 2, 3, \cdots.
\end{aligned}$$

2.3 Apply the method of Exercise 2.2 to obtain a third order Taylor series formula to approximate the solution to the IVP

$$\dot{y} = ty + 1,\ y(0) = 0. \tag{57}$$

Implement your formula by using a step size of $h = 0.25$ to approximate $y(1)$.

2.4 Approximate Dawson's integral, $e^{-t^2}\int_0^t e^{-s^2}ds$, on the interval $[0.0, 0.5]$ using the Taylor series algorithms of orders $p = 1, 2, 3$ with $h = 0.1$. Dawson's integral is the solution to the IVP,

$$\dot{y} = 1 - 2ty,\ y(0) = 0, \tag{58}$$

and its values at $t = 0.1, 0.2, 0.3, 0.4$, and 0.5, correct to five decimal places, appear in Table 7.

2.5 Consider the IVP

$$\dot{y} = -2y,\ y(0) = 1. \tag{59}$$

a) Solve the problem analytically to find $y(1)$.
b) Write a computer program that implements the Taylor series algorithm of order $p = 2$ for this problem. Let your program begin with a fixed step size $h = 1/n$ and repeatedly halve h

t	$y(t)$
0.1	0.09934
0.2	0.19475
0.3	0.28263
0.4	0.35994
0.5	0.42444

Table 7: Dawsons integral.

until a given accuracy, say $10^{-\alpha}$, is achieved; i.e., continue to halve until $|y(1) - y_n| < 10^{-\alpha}$. Let $\alpha = 3$ and start with $n = 2$.
c) Repeat (b) using Taylor algorithms of orders $p = 3$ and $p = 4$ and note the difference in the final values as p increases for fixed n. This example points out that the Taylor algorithms can be quite effective when the derivatives are easy to evaluate.

2.6 (Optional) Use MAPLE or MATHEMATICA (or a comparable symbolic package) to derive Taylor formulas of the indicated order p for the following IVPs:

$$\begin{aligned} a)\ \dot{y} &= 1 - 2ty,\ y(0) = 0;\ p = 4. \\ b)\ \dot{y} &= 2ty^2,\ y(0) = 0.5;\ p = 3. \\ c)\ \dot{y} &= \cos t \sin y,\ y(0) = 1;\ p = 3. \end{aligned}$$

Exercises to Illustrate Runge-Kutta Methods

— Exercise Set 3

3.1 Derive the expansion (32) in the text (Hint: Proceed by a succession of one-variable expansions, e.g.,

$$\begin{aligned} f(t+\Delta, y+\delta) &= f(t, y+\delta) + f_t(t, y+\delta)\Delta + \cdots \\ &= f(t,y) + f_y(t,y)\delta + f_t(t,y)\Delta + f_{ty}(t,y)\Delta\delta + \cdots . \end{aligned}$$

3.2 Write out the system of equations (36) as in Example 11 for each of the following IVPs.

$a)$ $\dot{Y}_1 = Y_2,\ \dot{Y}_2 = Y_1,\ Y_1(0) = 1,\ Y_2(0) = 0.$
$b)$ $\dot{Y}_1 = -4Y_1 - Y_2,\ \dot{Y}_2 = Y_1 - 2Y_2,\ Y_1(0) = 0,\ Y_2(0) = -1.$
$c)$ $\ddot{y} + t\dot{y} = \sin t,\ y(0) = 0,\ \dot{y}(0) = -2.$
$d)$ $\ddot{y} + y = \ln t,\ y(1) = 0,\ \dot{y}(1) = -1.$

3.3 Approximate the solution to the following IVPs at $t = 0.3$ using the Euler-Cauchy method with step size $h = 0.1$. Do your calculations by hand and round all results to four decimal places. Compare your numerical results with the true solutions at $t = 0.1, 0.2$, and 0.3

$a)$ $\dot{y} = y$, $y(0) = 1$.
$b)$ $\dot{y} = t - y$, $y(0) = 0$.
$c)$ $\dot{y} = 1 + y^2$, $y(0) = 0$.
$d)$ $\dot{y} = -y + t + 4$, $y(0) = 2$.
$e)$ $\dot{y} = \frac{4t}{y} - ty$, $y(0) = 3$.
$f)$ $\ddot{y} - 3\dot{y} + 2y = e^{-t}$, $y(1) = 0$, $\dot{y}(1) = 0$.
$g)$ $\ddot{y} + 4\dot{y} + 8y = \sin t$, $y(0) = 1$, $\dot{y}(0) = 0$.
$h)$ $\ddot{y} + y = \sin t$, $y(0) = 0$, $\dot{y}(0) = 2$.

3.4 Write a computer program to use the Euler method (29), the Euler-Cauchy method (34) with $\gamma = 1$, and the classical Runge-Kutta method (35) to approximate the solutions to the IVPs in Exercise 3.3 at $t = 0.1, 0.2$, and 0.3. Compare your numerical results with the exact solution at the indicated t points.

3.5 Write a computer program to use the classical Runge-Kutta method (35) to approximate the solution to the IVP in Example 11 at $t = 0.1, 0.2, \ldots, 1.0$. Use a step size $h = 0.1$. Compare your numerical results with the exact solution at the indicated t points.

3.6 Consider the problem,

$$\dot{y} = 2|t|y, \; y(-1) = 1/e, \tag{60}$$

on the interval $[-1, 1]$. Find the analytical solution $y(t)$ and show that $\dot{y}(t)$ is continuous on $[-1, 1]$, but $\ddot{y}(t)$ is not. Study the behavior of the error in Euler's method at some fixed points on $[-1, 1]$ as $h \to 0$.

3.7 When $f(t, y)$ depends only on t, show that the classical fourth order Runge-Kutta formula (35) reduces to Simpson's rule

$$\int_a^{a+h} f(t)dt = \frac{h}{6}[f(a) + 4f(a + \frac{h}{2}) + f(a + h)]. \tag{61}$$

What is the order of Simpsons's rule; i.e., what is the highest degree polynomial $P(t)$ that the rule integrates exactly? To what quadrature rule does the Runge-Kutta method of order 2 correspond when $\gamma = \frac{1}{2}$? When $\gamma = 1$?

Exercises Using the Code RKSUITE

— Exercise Set 4

4.1 To explore the various options in RKSUITE, compile and run the 6 codes contained in the file "templates;" the file appears in compressed form as "templates.Z." The file templates also contains output from the 6 codes, including data for all solution pairs: (2,3), (4,5), and (7,8).

4.2 Using the classical Runge-Kutta method (35), approximate $y(b)$ in each of the following IVPs. Choose the step size h to achieve a relative error of 10^{-6}, i.e., 6 significant digits of accuracy. The true solutions are given in each case for comparison.

$$\begin{aligned}
a)\ \dot{y} &= -\frac{1}{2}y^3,\ y(0)=1,\ b=3;\ y(t)=\frac{1}{\sqrt{1+t}}.\\
b)\ \dot{y} &= -2ty^2,\ y(0)=1,\ b=1;\ y(t)=\frac{1}{1+t^2}.\\
c)\ \dot{y} &= \frac{1}{4}y(1-\frac{y}{20}),\ y(0)=1,\ b=5;\ y(t)=\frac{20}{1+19e^{-t/4}}.\\
d)\ \dot{y} &= 100(\sin t - y),\ y(0)=0,\ b=1;\\
& \quad y(t)=\frac{10^2(e^{-100}t-\cos t)+10^4\sin t}{10^4+1}.\\
e)\ \dot{y} &= \frac{15\cos t}{y},\ y(0)=2,\ b=\frac{\pi}{4};\ y(t)=\sqrt{3\sin 10t+4}.
\end{aligned}$$

4.3 Repeat Exercise 4.2 using RKSUITE with TOL= 10^{-6}, THRES(1)= 1, and METHOD= 2. Are the actual errors within the requested tolerance? Which problems needed the most time, i.e., required the most number of function evaluations? Why? How does the amount of work compare in each case with that of using the classical Runge-Kutta formula (See Exercise 4.2)?

4.4 Use RKSUITE to approximate the solution to the sample problem in Sec. 2.5 for values of $\varepsilon \neq 1$, say $\varepsilon = 0.5, 5.0, 10.0, 100.0$, etc. Use "xmgr" to display your results in the phase plane for each value of ε. What happens as ε becomes "large"?

4.5 Use RKSUITE to approximate the solution to the following IVPs:

$$\begin{aligned}
a)\ \dot{y} &= 1+y^2,\ y(0)=0,\ 0\le t\le 2.\\
b)\ \dot{y} &= \sqrt{|1-y^2|},\ y(0)=1,\ 0\le t\le 10.\\
c)\ \dot{y} &= \sqrt{|1-y^2|},\ y(0)=1+\varepsilon,\ 0\le t\le 10.
\end{aligned}$$

where ε is on the order of unit roundoff on your machine. Contrast your result with what you obtained in b.

4.6 Consider the IVP

$$y^{(3)} - \ddot{y}\sin t - 2\dot{y}\cos t + y\sin t = \log t,$$
$$y(1) = A_1,\ \dot{y}(1) = A_2,\ \ddot{y}(1) = A_3.$$

Show that the solution, $y(t)$, satisfies the two integral relations:

$$\ddot{y}(t) - \dot{y}(t)\sin t - y(t)\cos t = c_2 + t\ln t - t \tag{62}$$

and

$$\dot{y}(t) - y(t)\sin t = c_1 + c_2 t + \frac{1}{2}t^2\ln t - \frac{3}{4}t^2. \tag{63}$$

Find c_1, c_2 in terms of A_1, A_2, and A_3. Use RKSUITE to approximate the solution to this problem and monitor the accuracy of your solution by seeing how well the integral relations are satisfied. Note that if the integral relations are satisfied, the numerical solution may or may not be accurate, but if they are not satisfied, the numerical solution must be inaccurate.

4.7 Use RKSUITE to approximate the solution to the IVP

$$\dot{P}(t) = 10^{-3}P(t)\left[10^3(1 - 0.3\cos\frac{\pi t}{6}) - P(t)\right],\ P(0) = 250. \tag{64}$$

Use "xmgr" to plot $P(t)$ vs. t for $0 \le t \le 36$.

4.8 The response of a motor controlled by a governor can be modeled by the following system of differential equations:

$$\begin{aligned}
\ddot{s} + 0.042\dot{s} + 0.961s &= \dot{\theta} + 0.063\theta,\\
\ddot{u} + 0.087\dot{u} &= \dot{s} + 0.025s,\\
\dot{v} &= 0.873(u - v),\\
\dot{w} &= 0.433(v - w),\\
\dot{x} &= 0.508(w - x),\\
\dot{\theta} &= -0.396(x - 47.6).
\end{aligned}$$

The motor should approach a constant (steady state) speed as $t \to \infty$. Assume that $s(0) = \dot{s}(0) = \dot{u}(0) = \theta(0) = 0$, $u(0) = 50$, and $v(0) = w(0) = x(0) = 75$. Evaluate $v(t)$ for $t = 0, 25, 50, 75, 100, 150, 200, 250, 300, 400, 500$. What does $\lim_{t\to\infty} v(t)$ appear to be? You can check this by calculating the exact steady state solution which occurs when all derivatives are zero.

4.9 A simple model of the human heart beat gives

$$\begin{aligned}
\varepsilon\dot{x} &= -(x^3 - Ax + c),\\
\dot{c} &= x,
\end{aligned}$$

where $x(t)$ is the displacement from equilibrium of the muscle fiber, $c(t)$ is the concentration of a chemical control, ε and A are positive constants. Assume $\varepsilon = 1.0$, $A = 3$.

a) Calculate $x(t)$ and $c(t)$ for $0 \le t \le 12$ starting with $x(0) = 0.1$, $c(0) = 0.1$. Use "xmgr" to plot the output in the phase plane (x along the horizontal axis, c on the vertical). What does the period appear to be?
b) Repeat with $x(0) = 0.87$, $c(0) = 2.1$.

4.10 In modeling circuits containing devices whose electrical properties are current dependent, ODEs of the form

$$A(\mathbf{X})\dot{\mathbf{X}} = \mathbf{F}(t, \mathbf{X}) \tag{65}$$

occur. For the case where

$$A(\mathbf{X}) = \begin{bmatrix} 3 - \frac{X_1^2}{4} & 0 & 1 - \frac{X_3^2}{8} \\ 0 & 4 & 1 - \frac{X_3^2}{8} \\ 3 - \frac{X_1^2}{4} & 2 & 4 - \frac{X_3^2}{8} \end{bmatrix},$$

$$\mathbf{F}(t, \mathbf{X}) = \begin{bmatrix} 30\cos t - 4X_1 + 5X_3 \\ 2X_1 - 3X_2 \\ 3X_2 - 3X_3 \end{bmatrix},$$

$$\mathbf{X}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

compute $\mathbf{X}(t)$, $t = 0.4, 0.8, 1.2, \ldots, 16$. Plot $X_1, X_2(t), X_3(t)$, over the interval $0 \le t \le 16$ on separate graphs using "xmgr." Use RKSUITE in conjunction with a linear system solver.

4.11 The orbit of the planet Mercury around the Sun can be represented as the solution to the differential equation,

$$\frac{d^2u}{d\theta^2} + u = \frac{\mu}{h^2}(1 + \varepsilon u^2), \tag{66}$$

where $u = 1/r$ and r denotes the distance from the Sun to Mercury. Here θ is an angle in the plane of the orbit, μ is the gravitational constant, h is the angular momentum, and ε is a parameter determined by the effects of other planets on Mercury as well as the Sun's oblateness, and a correction required by the general theory of relativity. To solve this problem, we convert it to the first order system

$$\begin{aligned} \frac{dv}{d\theta} &= w, \\ \frac{dw}{d\theta} &= 1 - v + \gamma v^2, \end{aligned}$$

where $v^2 = h^2(u/\mu)$ and $\gamma = \varepsilon(\mu^2/h^4)$.

To illustrate the phenomenon of *precession*, choose $\gamma = 0$, $v(0) = 2$, $w(0) = 0$, $\varepsilon = 0.01$, and integrate the system over several revolutions. Plot $y = (1/v)\sin\theta$ vs. $x = (1/v)\cos\theta$ using "xmgr". The plot should show that Mercury moves on an ellipse that is slowly rotating in the orbital plane. The points of closest approach to the Sun are called *perihelia*; the precession of these points is due to the perturbing nonlinearity in the differential equation. The observed precession of the perihelion of Mercury could not be explained by Newtonian mechanics and remained a puzzle for many years. The closest agreement between observations and the orbit modeled by the differential equation with ε containing a relativistic correction is one of the major experimental confirmations of Einstein's theory of general relativity.

4.12 This exercise illustrates the shooting algorithm for solving

$$\ddot{y} + p(t)\dot{y} + q(t)y = r(t),\ y(a) = A,\ y(b) = B. \tag{67}$$

Geometrically, we seek a function $y(t)$ that satisfies the differential equation and whose graph passes through the points (a,A) and (b,B). Our approach is to determine $\dot{y}(a)$, then we would have an initial value problem and RKSUITE could be used to solve it. So, let $\dot{y}(a) = s$ and the task is to find $s = s^*$ so that the resulting solution, denoted by $Y(t; s^*)$, satisfies $Y(b; s^*) = B$. We seek a zero of the function,

$$u(s) = Y(b; s) - B. \tag{68}$$

Step 1: Choose $s = s_1$ and solve the differential equation with initial conditions $y(a) = A_1$ and $\dot{y}(a) = s_1$; denote the resulting solutions by $Y(t; s_1)$. If

$$u(s_1) \quad = \quad Y(b; s_1) - B \quad = \quad 0,$$

set $s^* = s_1$ and stop. The solution is $y(t) = Y(t; s^*)$.
Step 2: Choose $s = s_2 \neq s_1$ and solve the differential equation with $y(a) = A$ and $\dot{y}(a) = s_2$; denote the solution by $Y(t; s_2)$. If

$$u(s_2) \quad = \quad Y(b; s_2) - B \quad = \quad 0,$$

set $s^* = s_2$ and stop. The solution is $y(t) = Y(t; s^*)$.
Step 3. Calculate the values of s for which $u(s) = 0$:

$$s^* = s_1 - \frac{s_1 - s_2}{u(s_1) - u(s_2)} u(s_1). \tag{69}$$

Note that, since our differential equation is linear, $u(s)$ is a linear function of s.
Step 4. Solve the differential equations with $y(a) = a$, $\dot{y}(a) = s^*$ to get the desired solution. The process is illustrated graphically in Figure 6.

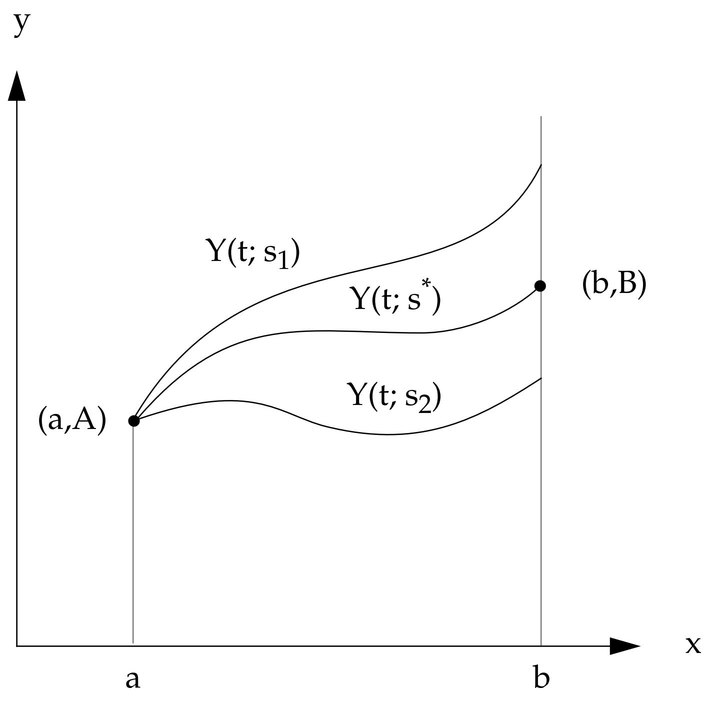

Figure 8: Graphic illustration of the shooting algorithm.

Use the shooting method to solve the following boundary value problems. Plot your solutions:

$$\begin{aligned}
a) &\quad \ddot{y} + \frac{2}{t}\dot{y} + y \;=\; 0,\ y(1) = 1,\ y(2) = 5,\ \text{answer } y(1) = 10.241848.\\
b) &\quad \ddot{y} + \dot{y} + ty = 0,\ y(0) = 1,\ y(1) = 0.\\
c) &\quad \ddot{y} + \frac{2}{t}\dot{y} - \frac{2}{t^2}y = \frac{\sin t}{t^2},\ y(1) = 1,\ y(2) = 2.\\
d) &\quad \ddot{y} + 4y = \cos t,\ y(0) = 0, y(\frac{\pi}{4}) = 0.\\
e) &\quad \ddot{y} + \sqrt{t}\dot{y} + y = e^t,\ y(0) = 0,\ y(1) = 0.\\
f) &\quad \ddot{y} + 4ty + (4t^2 + 2)y = 0,\ y(0) = 0,\ y(1) = \frac{1}{e}.\\
g) &\quad \ddot{y} - \frac{2t}{1-t^2}\dot{y} + \frac{12}{1-t^2}y = 0,\ y(0) = 0,\ y(0.5) = 4.
\end{aligned}$$

4.13 Modify the algorithm in Exercise 4.11 to solve the problem

$$(t^2 + 1)\ddot{y} - 2y = 0,\ y(0) \;=\; 1,\ \dot{y}(1) \;=\; 2. \tag{70}$$

4.14 Explain why the algorithm in Exercise 4.11 cannot be used to solve the boundary value problem

$$\ddot{y} + e^{-y} = 0,\ y(0) \;=\; 0,\ y(1) \;=\; 0. \tag{71}$$

Modify the algorithm to obtain an approximate solution to this problem.

4.15 The following differential equations arise in semi-conductor theory:

$$\begin{aligned}\varepsilon^2 \dot{E} &= p - n + 1,\\ \varepsilon \dot{p} &= pE - \varepsilon,\\ \varepsilon \dot{n} &= -nE + \varepsilon.\end{aligned}$$

Typical side conditions are $n(0) = p(0)$, $p(1) = 0$, $n(1) = 1$. For $\varepsilon = 1$ find $E(1)$ such that $n(0) = p(0)$. Print out your final value for $E(1)$. What happens if $\varepsilon = 10^{-2}$?

Exercises to Illustrate the Code VODE

— Exercise Set 5

5.1 Use the code VODE with MF = 21 (BDF method) to solve the IVP

$$\begin{aligned}&\dot{y} = \lambda(y - F(t)) + \dot{F}(t), \qquad 0 \le t \le 10,\\ &y(0) = 1,\end{aligned}$$

for $\lambda = -10, -20, -30, -100$, $F(t) = t$. Compare the amount of work that the code must do (number of function evaluations) with that required by RKSUITE; use relative and absolute error tolerances of 10^{-6}. Plot your solution for the case $\lambda = -10$.

5.2 Consider the van der Pol equation in relaxed oscillation:

$$\begin{aligned}&\ddot{y} + 1000(y^2 - 1)\dot{y} + y, \qquad 0 \le t \le 3000,\\ &y(0) = 2,\ \dot{y}(0) = 0.\end{aligned}$$

The initial conditions are close to a limit cycle, and, in particular, to a slowly varying part of the cycle. This strains step size selection algorithms because the solution is so easy to approximate. With the parameter of 1000 the oscillations exhibits regions of very sharp change where the step size must be small to resolve the solution and the problem is not stiff. Elsewhere, the solution varies slowly and the problem is stiff; the problem is very stable here and all nontrivial solutions tend to vary rapidly to the limit cycle. The limit cycle has a period of about 1615.5 and the interval has been chosen to exhibit several of the internal boundary layers (regions of sharp change).
a) Use VODE with MF = 21 (BDF method) to approximate the solution on $0 \le t \le 3000$ and generate the plots y .vs. t, $\dot{y}$ .vs. t, and $\dot{y}$ .vs. y (phase plane). Identify the intervals over which the problem appears to be stiff.
b) Use VODE with MF = 10 (Adams-Moulton method) to approximate the solution on $0 \le t \le 3000$. What happens?
c) Repeat with RKSUITE.
This problem is very hard for a code intended for non-stiff problems as it changes type from stiff to non-stiff and back as the integration proceeds.

Miscellaneous Exercises

— Exercise Set 6

6.1 The following system has been used to study chemicals that are active in the atmosphere ozone depletion cycle [14, pp 317]. It involves the rapid formation of chlorine atoms from oxygen and hydrogen atoms and molecular chlorine:

$$\begin{array}{rclr}
Cl + H_2 & \xrightarrow{k_1} & HCl + H & k_1 = 1.6 \times 10^{-14}; \\
H + Cl_2 & \xrightarrow{k_2} & HCl + Cl & k_2 = 1.0 \times 10^{-11}; \\
H + O_2 & \xrightarrow{k_3} & HO_2 & k_3 = 3.6 \times 10^{-13}; \\
Cl + O_2 & \xrightarrow{k_4} & ClO_2 & k_4 = 1.3 \times 10^{-14}; \\
Cl + ClO_2 & \xrightarrow{k_5} & Cl_2 + O_2 & k_5 = 1.4 \times 10^{-10}.
\end{array}$$

There is a wide range of time constants in this problem that is typical of stiff ODEs. The numerical stability of most methods will be limited by k_1 and k_4 relative to the method's step size. Using the labeling

$$\begin{array}{ll}
Y_1 = Cl, & Y_2 = H_2, \\
Y_3 = Cl_2, & Y_4 = HCl, \\
Y_5 = H, & Y_6 = O_2, \\
Y_7 = HO_2, & Y_8 = ClO_2,
\end{array}$$

we arrive at the following set of ODE's:

$$\begin{array}{rcl}
\dot{Y}_1 & = & -k_1 Y_1 Y_2 + k_2 Y_3 Y_5 - k_4 Y_1 Y_6 - k_5 Y_1 Y_8, \\
\dot{Y}_2 & = & -k_1 Y_1 Y_2, \\
\dot{Y}_3 & = & -k_2 Y_3 Y_5 + k_5 Y_1 Y_8, \\
\dot{Y}_4 & = & k_1 Y_1 Y_2 + k_2 Y_3 Y_5, \\
\dot{Y}_5 & = & k_1 Y_1 Y_2 - k_2 Y_3 Y_5 - k_3 Y_5 Y_6, \\
\dot{Y}_6 & = & -k_3 Y_5 Y_6 - k_4 Y_1 Y_6 + k_5 Y_1 Y_8, \\
\dot{Y}_7 & = & k_3 Y_5 Y_6, \\
\dot{Y}_8 & = & k_4 Y_1 Y_6 - k_5 Y_1 Y_8.
\end{array}$$

For the initial conditions,

$$\begin{array}{rcl}
Y_1(0) & = & 1.00 \times 10^{14}, \\
Y_2(0) & = & 1.62 \times 10^{18}, \\
Y_3(0) & = & 3.25 \times 10^{16},
\end{array}$$

$$\begin{aligned}
Y_4(0) &= 0.0, \\
Y_5(0) &= 0.0, \\
Y_6(0) &= 4.84 \times 10^{18}, \\
Y_7(0) &= 0.0, \\
Y_8(0) &= 0.0,
\end{aligned}$$

tabulate the solution at 2×10^{-4} using both the Adams method (VODE with MF=10) and the BDF method (VODE with MF=22, internally generated Jacobian). Obtain the elapsed time for each solution. This problem would be classified as mildly stiff and you should notice a considerable difference in the elapsed solution times. For error tolerances use, ITOL=1, RTOL=10^{-3}, and ATOL=10^{-6}.

6.2 Use an appropriate integration method (e.g., improved Euler) to determine the qualitative behavior of each of the following dynamical systems. Plot your solutions.

a) Lorentz system:

$$\begin{aligned}
\dot{x} &= \sigma(y - x), \\
\dot{y} &= x(\rho - z) - y, \\
\dot{z} &= xy - bz,
\end{aligned}$$

on the interval $0 \leq t \leq 200$ with constants

$$\sigma = 10.0, \quad \rho = 28.0, \quad b = 8/3.$$

Use the initial conditions

$$x(0) = -1.3560, \quad y(0) = -2.492152, \quad z(0) = 12.317410.$$

b) Francheschini system:

$$\begin{aligned}
\dot{y}_1 &= -p_1 y_1 + p_2 y_2 y_3 + p_2 y_4 y_5, \\
\dot{y}_2 &= -p_3 y_2 + p_4 y_1 y_3 + p_4 y_6 y_7, \\
\dot{y}_3 &= -p_5 y_3 + p_6 y_1 y_7 - p_7 y_1 y_2 + R, \\
\dot{y}_4 &= -p_5 y_4 - p_9 y_1 y_5, \\
\dot{y}_5 &= -y_5 - p_4 y_1 y_4, \\
\dot{y}_6 &= -p_8 y_6 - p_2 y_2 y_7, \\
\dot{y}_7 &= -p_5 y_7 + p_9 y_2 y_6 - p_6 y_1 y_3,
\end{aligned}$$

on the interval $0 \leq t \leq 5$, with p-parameter values of

$$\begin{array}{lll}
p_1 = 2.0, & p_2 = 8.94427, & p_3 = 9.0, \\
p_4 = 6.70820, & p_5 = 5.0, & p_6 = 9.0, \\
p_7 = 15.6524, & p_8 = 8.0, & p_9 = 2.23607,
\end{array}$$

and $R = 3.0$. Use the initial conditions

$$\begin{aligned}
y_1(0) &= -3.97427, \\
y_2(0) &= -0.338907, \\
y_3(0) &= 5.396108, \\
y_4(0) &= -1.29233, \\
y_5(0) &= 6.81998, \\
y_6(0) &= 0.625067, \\
y_7(0) &= 6.30711.
\end{aligned}$$

c) Rossler system:

$$\begin{aligned}
\dot{x} &= -(y + z), \\
\dot{y} &= x + ay, \\
\dot{z} &= b + z(x - c),
\end{aligned}$$

on the interval $0 \leq t \leq 200$ with constants

$$a = 1/5, \quad b = 1/5, \quad c = 5.7.$$

Use the initial conditions

$$x(0) = -2.209787, \quad y(0) = 1.353531, \quad z(0) = 0.070299.$$

References

[1] D.A. Sanchez, R.C. Allen, Jr., and W.T. Kyner, *Differential Equations.* Addison-Wesley. 1988.

[2] W.E. Boyce and R.C. DiPrima, *Elementary Differential Equations*, 4th ed, Wiley, New York, 1986.

[3] F. Brauer and J.A. Nohel, *Ordinary Equations with Applications*, Harper and Row, New York, 1986.

[4] L.F. Sampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, Freeman, 1975.

[5] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, 1971.

[6] L.F. Shampine and C.W. Gear, A Users view of Solving Stiff Ordinary Differential Equations, SIAM Review, 21 (1979) pp. 1-17.

[7] R.W. Brankin, I. Gladwell, and L.F. Shampine, RKSUITE: a Suite of Runge-Kutta Codes for the Initial Value Problem of ODEs. Softreport 92-S1, Department of Mathematics, Southern Methodist University, Dallas, TX, USA, 1992.

[8] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh, VODE: a Variable-coefficient ODE Solver, SIAM J.Sci. Stat. Comput., Vol. 10, No. 5, pp. 1038-1051, Sept. 1989.

[9] J. Butcher *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods* Wiley, 1987.

[10] G. Hall and J. Watt *Modern Numerical Methods for Ordinary Differential Equations*, Clarendon Press, 1976.

[11] J. Lambert, *Numerical Methods for Ordinary Differential Equations*, Wiley, 1991.

[12] J. Lambert, S.P. Norseth, and C. Warner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer-Verlag 1987.

[13] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer-Verlag, 1991.

[14] L.F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, 1994.

[15] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice-Hall, 1989.

[16] G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, 1977.

[17] L. Shampine and R.C. Allen, Jr. *Numerical Computing: An Introduction*, Saunders, 1973.

[18] M. Abramowitz and I.E. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Dover Publications, Inc., New York, 1977.

Acknowledgements

The author would like to express his appreciation to the following people for their assistance in the preparation of this chapter: Larry Shampine (Southern Methodist University), Tom Kyner (University of New Mexico), Kris Stewart (San Diego State University), and Dave Zachmann (Colorado State University). Text material and examples were taken freely from [1], Chapters 3 and 8.