Behind and Beyond the MATLAB ODE Suite*

Ryuichi Ashino† Michihiro Nagase‡
Rémi Vaillancourt§

CRM-2651

January 2000

*Dedicated to Professor Norio Shimakura on the occasion of his sixtieth birthday

†Division of Mathematical Sciences, Osaka Kyoiku University, Kashiwara, Osaka 582, Japan; ashino@cc.osaka-kyoiku.ac.jp

‡Department of Mathematics, Graduate School of Science, Osaka University, Toyonaka, Osaka 560, Japan nagase@math.wani.osaka-u.ac.jp

§Department of Mathematics and Statistics, University of Ottawa, Ottawa, Ontario, Canada K1N 6N5 remiv@mathstat.uottawa.ca

Abstract

The paper explains the concepts of *order* and *absolute stability* of numerical methods for solving systems of first-order ordinary differential equations (ODE) of the form

$$y' = f(t, y), \quad y(t_0) = y_0, \quad \text{where } f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n,$$

describes the phenomenon of problem *stiffness*, and reviews explicit Runge–Kutta methods, and explicit and implicit linear multistep methods. It surveys the five numerical methods contained in the MATLAB ODE suite (three for nonstiff problems and two for stiff problems) to solve the above system, lists the available options, and uses the `odedemo` command to demonstrate the methods. One stiff ode code in MATLAB can solve more general equations of the form $M(t)y' = f(t, y)$ provided the `Mass` option is `on`.

Keywords : stiff and nonstiff differential equations, implicit and explicit ODE solvers, Matlab odedemo

Résumé

On explique les concepts d'*ordre* et de *stabilité absolue* d'une méthode numérique pour résoudre le système d'équations différentielle du premier ordre :

$$y' = f(t, y), \quad y(t_0) = y_0, \quad \text{où } f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n,$$

et le phénomène des problèmes *raides*. On décrit les méthodes explicites du type Runge–Kutta et les méthodes multipas linéaires explicites et implicites. On décrit les cinq méthodes de la suite ODE de MATLAB, trois pour les problèmes non raides et deux pour les problèmes raides. On dresse la liste des options disponibles et on emploie la commande `odedemo` pour illustrer les méthodes. Un des codes de MATLAB peut résoudre des systèmes plus généraux de la forme $M(t)y' = f(t, y)$ si l'on active l'option `Mass`.

1 Introduction

The MATLAB ODE suite is a collection of five user-friendly finite-difference codes for solving initial value problems given by first-order systems of ordinary differential equations and plotting their numerical solutions. The three codes `ode23`, `ode45`, and `ode113` are designed to solve non-stiff problems and the two codes `ode23s` and `ode15s` are designed to solve both stiff and non-stiff problems. The purpose of this paper is to explain some of the mathematical background built in any finite difference methods for accurately and stably solving ODE's. A survey of the five methods of the ODE suite is presented. As a first example, the van de Pol equation is solved by the classic four-stage Runge–Kutta method and by the MATLAB `ode23` code. A second example illustrates the performance of the five methods on a system with small and with large stiffness ratio. The available options in the MATLAB codes are listed. The 19 problems solved by the MATLAB `odedemo` are briefly described. These standard problems, which are found in the literature, have been designed to test ode solvers.

2 Initial Value Problems

Consider the initial value problem for a system of n ordinary differential equations of first order:

$$y' = f(t, y), \qquad y(t_0) = y_0, \tag{1}$$

on the interval $[a, b]$ where the function $f(t, y)$:

$$f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n,$$

is continuous in t and Lispschitz continuous in y, that is,

$$\|f(t, y) - f(t, x)\| < M\|y - x\|, \tag{2}$$

for some positive constant M. Under these conditions, problem (1) admits one and only one solution, $y(t)$, $t \in [a, b]$.

There are many numerical methods in use to solve (1). Methods may be explicit or implicit, one-step or multistep. Before we describe classes of finite-difference methods in some detail in Section 6, we immediately recall two such methods to orient the reader and to introduce some notation.

The simplest explicit method is the one-step Euler method:

$$y_{n+1} = y_n + h_n f(t_n, y_n), \qquad t_{n+1} = t_n + h_n, \qquad n = 0, 1, \ldots,$$

where h_n is the step size and y_n is the numerical solution. The simplest implicit method is the one-step backward Euler method:

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1}), \qquad t_{n+1} = t_n + h_n, \qquad n = 0, 1, \ldots.$$

This last equation is usually solved for y_{n+1} by Newton's or a simplified Newton method which involves the Jacobian matrix $J = \partial_y f(t, y)$. It will be seen that when J is constant, one needs only set the option `JConstant` to `on` to tell the MATLAB solver to evaluate J only once at the start. This represents a considerable saving in time. In the sequel, for simplicity, we shall write h for h_n, bearing in mind that the codes of the ODE suite use a variable step size whose length is controlled by the code. We shall also use the shortened notation $f_n := f(t_n, y_n)$.

The Euler and backward Euler methods are simple but not very accurate and may require a very small step size. More accurate finite difference methods have developed from Euler's method in two streams:

- Linear multistep methods combine values $y_{n+1}, y_n, y_{n-1}, \ldots$, and $f_{n+1}, f_n, f_{n-1}, \ldots$, in a linear way to achieve higher accuracy, but sacrifice the one-step format. Linearity allows for a simple local error estimate, but makes it difficult to change step size.

- Runge–Kutta methods achieve higher accuracy by retaining the one-step form but sacrificing linearity. One-step form makes it easy to change step size but makes it difficult to estimate the local error.

An ode solver needs to produce a numerical solution y_n of system (1) which converges to the exact solution $y(t)$ as $h \downarrow 0$ with $hn = t$, for each $t \in [a, b]$, and remains stable at working step size. These questions will be addressed in Sections 3 and 4. If stability of an explicit method requires an unduly small step size, we say that the problem is stiff and use an implicit method. Stiffness will be addressed in Section 5.

3 Convergent Numerical Methods

The numerical methods considered in this paper can be written in the general form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\varphi_f(y_{n+k}, y_{n+k-1}, \ldots, y_n, t_n; h). \tag{3}$$

where the subscript f to φ indicates the dependence of φ on the function $f(t, y)$ of (1). We impose the condition that

$$\varphi_{f\equiv 0}(y_{n+k}, y_{n+k-1}, \ldots, y_n, t_n; h) \equiv 0,$$

and note that the Lipschitz continuity of φ with respect to y_{n+j}, $j = 0, 1, \ldots, k$, follows from the Lipschitz continuity (2) of f.

Definition 1. Method (3) with appropriate starting values is said to be **convergent** if, for all initial value problems (1), we have

$$y_n - y(t_n) \to 0 \quad \text{as} \quad h \downarrow 0,$$

where $nh = t$ for all $t \in [a, b]$.

The **local truncation error** of (3) is the residual

$$R_{n+k} := \sum_{j=0}^{k} \alpha_j y(t_{n+j}) - h\varphi_f(y(t_{n+k}), y(t_{n+k-1}), \ldots, y(t_n), t_n; h). \tag{4}$$

Definition 2. Method (3) with appropriate starting values is said to be **consistent** if, for all initial value problems (1), we have

$$\frac{1}{h} R_{n+k} \to 0 \quad \text{as} \quad h \downarrow 0,$$

where $nh = t$ for all $t \in [a, b]$.

Definition 3. Method (3) is **zero-stable** if the roots of the characteristic polynomial

$$\sum_{j=0}^{k} \alpha_j r^{n+j}$$

lie inside or on the boundary of the unit disk, and those on the unit circle are simple.

We finally can state the following fundamental theorem.

Theorem 1. A method is **convergent** as $h \downarrow 0$ if and only if it is zero-stable and consistent.

All numerical methods considered in this work are convergent.

4 Absolutely Stable Numerical Methods

We now turn attention to the application of a consistent and zero-stable numerical solver with small but nonvanishing step size.

For $n = 0, 1, 2, \ldots$, let y_n be the numerical solution of (1) at $t = t_n$, and $y^{[n]}(t_{n+1})$ be the exact solution of the **local** problem:

$$y' = f(t, y), \qquad y(t_n) = y_n. \tag{5}$$

A numerical method is said to have **local error**:

$$\varepsilon_{n+1} = y_{n+1} - y^{[n]}(t_{n+1}). \tag{6}$$

If we assume that $y(t) \in C^{p+1}[t_0, t_f]$, we have

$$\varepsilon_{n+1} \approx C_{p+1} h_{n+1}^{p+1} y^{(p+1)}(t_n) + O(h_{n+1}^{p+2}) \tag{7}$$

and say that C_{p+1} is the error constant of the method. For consistent and zero-stable methods, the global error is of order p whenever the local error is of order $p+1$. We remark that a method of order $p \geq 1$ is consistent according to Definition 2.

Let us now apply the solver (3), with its small nonvanishing parameter h, to the linear test equation

$$y' = \lambda y, \qquad \Re\lambda < 0. \tag{8}$$

The **region of absolute stability**, R, is that region in the complex $\hat{h}$-plane, where $\hat{h} = h\lambda$, for which the numerical solution y_n of (8) goes to zero, as n goes to infinity.

The region of absolute stability of the explicit Euler method is the disk of radius 1 and center $(-1, 0)$, see curve $s = 1$ in Fig. 1. The region of stability of the implicit backward Euler method is the outside of the disk of radius 1 and center $(1, 0)$, hence it contains the left half-plane, see curve $k = 1$ in Fig. 4.

The region of absolute stability, R, of an explicit method is very roughly a disk or cardioid in the left half-plane (the cardioid overlaps with the right half-plane with cusp at the origin), see Figs. 1, 2, and 3 . The boundary of R cuts the real axis at α, where $-\infty < \alpha < 0$, and at the origin. The interval $[\alpha, 0]$ is called the interval of absolute stability. For methods with real coefficients, R is symmetric with respect to the real axis. All methods considered in this work have real coefficients; hence figures show only the upper half of R.

The region of stability, R, of implicit methods extends to infinity in the left half-plane, that is $\alpha = -\infty$. The angle subtended at the origin by R in the left half-plane is usually smaller for higher order methods, see Fig. 4.

If the region R does not include the whole negative real axis, that is, $-\infty < \alpha < 0$, then the inclusion

$$h\lambda \in R$$

restricts the step size:

$$\alpha \leq h\,\Re\lambda \implies 0 < h \leq \frac{\alpha}{\Re\lambda}.$$

In practice, we want to use a step size h small enough to ensure accuracy of the numerical solution as implied by (6)–(7), but not too small.

5 The Phenomenon of Stiffness

While the intuitive meaning of stiff is clear to all specialists, much controversy is going on about its correct mathematical definition. The most pragmatic opinion is also historically the first one: stiff equations are equations where certain implicit methods, in particular backward differentiation methods, perform much better than explicit ones (see [1], p. 1).

Given system (1), consider the $n \times n$ Jacobian matrix

$$J = \partial_y f(t, y) = \left(\frac{\partial f_i}{\partial y_j}\right), \quad i \downarrow 1, \ldots, n, \quad j \rightarrow 1, \ldots, n, \tag{9}$$

where Nagumo's matrix index notation has been used. We assume that the n eigenvalues $\lambda_1, \ldots, \lambda_n$ of the matrix J have negative real parts, $\Re\lambda_j < 0$, and are ordered as follows:

$$\Re\lambda_n \leq \cdots \leq \Re\lambda_2 \leq \Re\lambda_1 < 0. \tag{10}$$

The following definition occurs in discussing stiffness.

Definition 4. The **stiffness ratio** of the system $y' = f(t, y)$ is the positive number

$$r = \frac{\Re\lambda_n}{\Re\lambda_1}, \tag{11}$$

where the eigenvalues of the Jacobian matrix (9) of the system satisfy the relations (8).

The phenomenon of stiffness appears under various aspects (see [2], p. 217–221):

- A linear constant coefficient system is stiff if all of its eigenvalues have negative real parts and the stiffness ratio is large.
- Stiffness occurs when stability requirements, rather than those of accuracy, constrain the step length.

- Stiffness occurs when some components of the solution decay much more rapidly than others.
- A system is said to be stiff in a given interval I containing t if in I the neighboring solution curves approach the solution curve at a rate which is very large in comparison with the rate at which the solution varies in that interval.

A statement that we take as a definition of stiffness is one which merely relates what is observed happening in practice.

Definition 5. If a numerical method with a region of absolute stability, applied to a system of differential equation with any initial conditions, is forced to use in a certain interval I of integration a step size which is *excessively small* in relation to the smoothness of the exact solution in I, then the system is said to be **stiff** in I.

Explicit Runge–Kutta methods and predictor-corrector methods, which, in fact, are explicit pairs, cannot handle stiff systems in an economical way, if they can handle them at all. Implicit methods require the solution of nonlinear equations which are almost always solved by some form of Newton's method.

6 Numerical Methods for Initial Value Problems

6.1 Runge–Kutta Methods

Runge–Kutta methods are one-step multistage methods. As an example, we recall the (classic) four-stage Runge–Kutta method of order 4 given by its formula (left) and conveniently in the form of a Butcher tableau (right).

$$
\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \\
k_3 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right) \\
k_4 &= f\left(t_n + h, y_n + hk_3\right) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

	c	A			
k_1	0	0			
k_2	1/2	1/2	0		
k_3	1/2	0	1/2	0	
k_4	1	0	0	1	0
y_{n+1}	b^T	1/6	2/6	2/6	1/6

In a Butcher tableau, the components of the vector c are the increments of t_n and the entries of the matrix A are the multipliers of the approximate slopes which, after multiplication by the step size h, increment y_n. The components of the vector b are the weights in the combination of the intermediary values k_j. The left-most column of the tableau is added here for the reader's convenience.

There are stable s-stage explicit Runge-Kutta methods of order $p = s$ for $s = 1, 2, 3, 4$. The minimal number of stages of a stable explicit Runge-Kutta method of order 5 is 6.

Applying a Runge-Kutta method to the test equation,

$$y' = \lambda y, \qquad \Re\lambda < 0,$$

with solution $y(t) \to 0$ as $t \to \infty$, one obtains a one-step difference equation of the form

$$y_{n+1} = Q(\hat{h})y_n, \qquad \hat{h} = h\lambda,$$

where $Q(\hat{h})$ is the stability function of the method. We see that $y_n \to 0$ as $n \to \infty$ if and only if

$$|Q(\hat{h})| < 1, \tag{12}$$

and the method is **absolutely stable** for those values of $\hat{h}$ in the complex plane for which (12) hold; those values form the **region of absolute stability** of the method. It can be shown that the stability function of explicit s-stage Runge-Kutta methods of order $p = s$, $s = 1, 2, 3, 4$, is

$$R(\hat{h}) = \frac{y_{n+1}}{y_n} = 1 + \hat{h} + \frac{1}{2!}\hat{h}^2 + \cdots + \frac{1}{s!}\hat{h}^s.$$

The regions of absolute stability of s-stage explicit Runge–Kutta methods of order $k = s$, for $s = 1, 2, 3, 4$, are the interior of the closed regions whose upper halves are shown in the left part of Fig. 1.

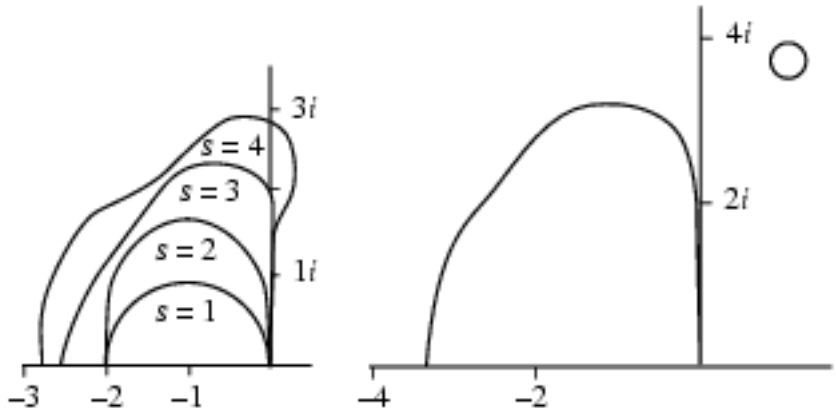

Figure 1: Left: Regions of absolute stability of s-stage explicit Runge–Kutta methods of order $k = s$. Right: Region of absolute stability of the Dormand-Prince pair DP5(4)7M.

Embedded pairs of Runge–Kutta methods of orders p and $p+1$ with interpolant are used to control the local error and interpolate the numerical solution between the nodes which are automatically chosen by the step-size control. The difference between the higher and lower order solutions, $y_{n+1} - \hat{y}_{n+1}$, is used to control the local error. A popular pair of methods of orders 5 and 4, respectively, with interpolant due to Dormand and Prince [3] is given in the form of a Butcher tableau.

Table 1: Butcher tableau of the Dormand-Prince pair DP5(4)7M with interpolant.

	c	A						
k_1	0	0						
k_2	$\frac{1}{5}$	$\frac{1}{5}$	0					
k_3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0				
k_4	$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	0			
k_5	$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	0		
k_6	1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	0	
k_7	1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
$\hat{y}_{n+1}$	$\hat{b}^T$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
y_{n+1}	b^T	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$y_{n+0.5}$		$\frac{5783653}{57600000}$	0	$\frac{466123}{1192500}$	$-\frac{41347}{1920000}$	$\frac{16122321}{339200000}$	$-\frac{7117}{20000}$	$\frac{183}{10000}$

The number 5 in the designation DP5(4)7M means that the solution is advanced with the solution y_{n+1} of order five (a procedure called *local extrapolation*). The number (4) in parentheses means that the solution $\hat{y}_{n+1}$ of order four is used to obtain the local error estimate. The number 7 means that the method has seven stages. The letter M means that the constant C_6 in the top-order error term has been minimized, while maintaining stability. Six stages are necessary for the method of order 5. The seventh stage is necessary to have an interpolant. However, this is really a six-stage method since the first step at t_{n+1} is the same as the last step at t_n, that is, $k_1^{[n+1]} = k_7^{[n]}$. Such methods are called FSAL (First Step As Last). The upper half of the region of absolute stability of the pair DP5(4)7M comprises the interior of the closed region in the left half-plane and the little round region in the right half-plane shown in the right part of Fig. 1.

Other popular pairs of embedded Runge-Kutta methods are the Runge–Kutta–Verner and Runge–Kutta–Fehlberg methods. For instance, the pair RKF45 of order four and five minimizes the error constant C_5 of the lower order method which is used to advance the solution from y_n to y_{n+1}, that is, without using local extrapolation.

One notices that the matrix A in the Butcher tableau of an explicit Rung–Kutta method is strictly lower triangular. Semi-explicit methods have a lower triangular matrix. Otherwise, the method is implicit. Solving semi-explicit methods for the vector solution y_{n+1} of a system is much cheaper than solving explicit methods.

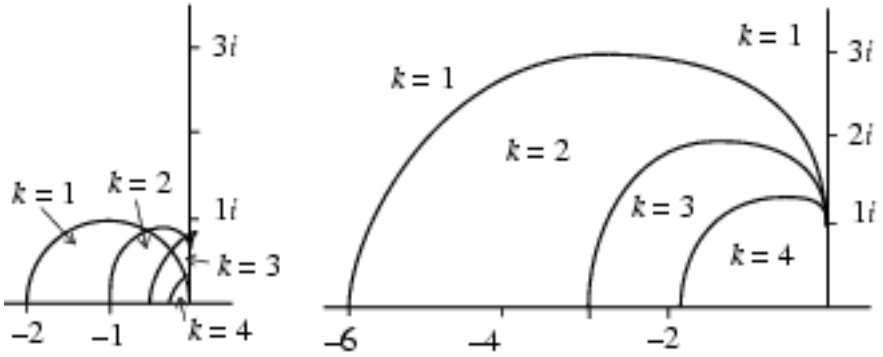

Figure 2: Left: Regions of absolute stability of k-step Adams–Bashforth methods. Right: Regions of absolute stability of k-step Adams–Moulton methods.

Runge–Kutta methods constitute a clever and sensible idea [2]. The unique solution of a well-posed initial value problem is a single curve in $\mathbb{R}^{n+1}$, but due to truncation and round-off error, any numerical solution is, in fact, going to wander off that integral curve, and the numerical solution is inevitably going to be affected by the behavior of neighboring curves. Thus, it is the behavior of the family of integral curves, and not just that of the unique solution curve, that is of importance. Runge–Kutta methods deliberately try to gather information about this family of curves, as it is most easily seen in the case of explicit Runge–Kutta methods.

6.2 Adams-Bashforth-Moulton Linear Multistep Methods

Using the shortened notation

$$f_n := f(t_n, y_n), \qquad n = 0, 1, \ldots,$$

we define a **linear multistep method** or **linear k-step method** in standard form by

$$\sum_{j=0}^{k} \alpha_j y_{n+j-k+1} = h \sum_{j=0}^{k} \beta_j f_{n+j-k+1}, \tag{13}$$

where α_j and β_j are constants subject to the normalizing conditions

$$\alpha_k = 1, \qquad |\alpha_0| + |\beta_0| \neq 0.$$

The method is explicit if $b_k = 0$, otherwise it is implicit.

Applying (13) to the test equation,

$$y' = \lambda y, \qquad \Re\lambda < 0,$$

with solution $y(t) \to 0$ as $t \to \infty$, one finds that the numerical solution $y_n \to 0$ as $n \to \infty$ if the zeros, $r_s(\hat{h})$, of the stability polynomial

$$\pi(r, \hat{h}) := \sum_{j=0}^{k} (\alpha_j - \hat{h}\beta_j) r^j$$

satisfy $|r_s(\hat{h})| < 1$, $s = 1, 2, \ldots, k$. In that case, we say that the linear multistep method (13) is **absolutely stable** for given $\hat{h}$. The **region of absolute stability**, R, in the complex plane is the set of values of $\hat{h}$ for with the method is absolutely stable. The regions of absolute stability of k-step Adams–Bashforth and Adams–Moulton methods of order $k = 1, 2, 3, 4$, are the interior of the closed regions whose upper halves are shown in the left and right parts, respectively, of Fig. 2. The region of absolute stability of the Adams–Bashforth method of order 3 extends in a small triangular region in the right half-plane. The region of absolute stability of the Adams–Moulton method of order 1 is the whole left half-plane.

Popular linear k-step methods are (explicit) Adams–Bashforth (AB) and (implicit) Adams–Moulton (AM) methods,

$$y_{n+1} - y_n = h \sum_{j=0}^{k-1} \beta_j^* f_{n+j-k+1}, \qquad y_{n+1} - y_n = h \sum_{j=0}^{k} \beta_j f_{n+j-k+1},$$

Table 2: Coefficients of Adams–Bashforth methods of stepnumber 1–6.

β_5^*	β_4^*	β_3^*	β_2^*	β_1^*	β_0^*	d	k	p	C_{p+1}^*
					1	1	1	1	1/2
				3	−1	2	2	2	5/12
			23	−16	5	12	3	3	3/8
		55	−59	37	−9	24	4	4	251/720
	1901	−2774	1616	−1274	251	720	5	5	95/288
4277	−7923	9982	−7298	2877	−475	1440	6	6	19 087/60 480

Table 3: Coefficients of Adams–Moulton methods of stepnumber 1–6.

β_5	β_4	β_3	β_2	β_1	β_0	d	k	p	C_{p+1}
				1	1	2	1	2	−1/12
			5	8	−1	12	2	3	−1/24
		9	19	−5	1	24	3	4	−19/720
	251	646	−264	106	−19	720	4	5	−3/160
475	1427	−798	482	−173	27	1440	5	6	−863/60 480

respectively. Tables 2 and 3 list the AB and AM methods of stepnumber 1 to 6, respectively. In the tables, the coefficients of the methods are to be divided by d, k is the stepnumber, p is the order, and C_{p+1}^* and C_{p+1} are the corresponding error constants of the methods.

In practice, an AB method is used as a **predictor** to predict the next-step value y_{n+1}^*. The function f is then evaluated as $f(x_{n+1}, y_{n+1}^*)$ and inserted in the right-hand side of an AM method used as a **corrector** to obtain the corrected value y_{n+1}. The function f is then evaluated as $f(x_{n+1}, y_{n+1})$. Such combination is called an ABM predictor-corrector in the PECE mode. If the predictor and corrector are of the same order, they come with the Milne estimate for the principal local truncation error

$$\epsilon_{n+1} \approx \frac{C_{p+1}}{C_{p+1}^* - C_{p+1}} (y_{n+1} - y_{n+1}^*).$$

This estimate can also be used to improve the corrected value y_{n+1}, a procedure that is called local extrapolation. Such combination is called an ABM predictor-corrector in the PECLE mode.

The regions of absolute stability of kth-order Adams–Bashforth–Moulton pairs, for $k = 1, 2, 3, 4$, in the PECE mode, are the interior of the closed regions whose upper halves are shown in the left part of Fig. 3. The regions of absolute stability of kth-order Adams–Bashforth–Moulton pairs, for $k = 1, 2, 3, 4$, in the PECLE mode, are the interior of the closed regions whose upper halves are shown in the right part of Fig. 3.

6.3 Backward Differentiation Formulas

We define a k-step **backward differentiation formula** (BDF) in standard form by

$$\sum_{j=0}^{k} \alpha_j y_{n+j-k+1} = h\beta_k f_{n+1},$$

where $\alpha_k = 1$. BDF's are implicit methods. Tables 4 lists the BDF's of stepnumber 1 to 6, respectively. In the table, k is the stepnumber, p is the order, C_{p+1} is the error constant, and α is half the angle subtended at the origin by the region of absolute stability R.

The left part of Fig. 4 shows the upper half of the region of absolute stability of the 1-step BDF, which is the exterior of the unit disk with center 1, and the regions of absolute stability of the 2- and 3-step BDF's which are the

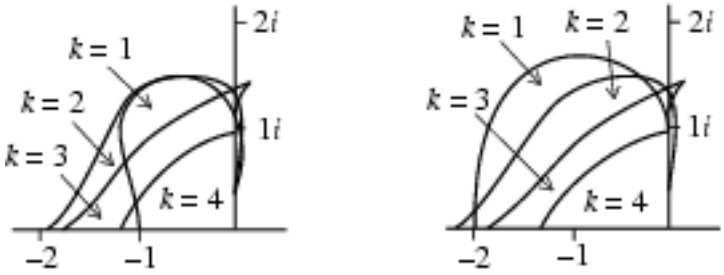

Figure 3: Regions of absolute stability of k-order Adams–Bashforth–Moulton methods, left in PECE mode, and right in PECLE mode.

Table 4: Coefficients of the BDF methods.

k	α_6	α_5	α_4	α_3	α_2	α_1	α_0	β_k	p	C_{p+1}	α
1						1	-1	1	1	1	90°
2					1	$-\frac{4}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	2	$-\frac{2}{9}$	90°
3				1	$-\frac{18}{11}$	$\frac{9}{11}$	$=\frac{2}{11}$	$\frac{6}{11}$	3	$-\frac{3}{22}$	86°
4			1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$	$\frac{12}{25}$	4	$-\frac{12}{125}$	73°
5		1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	$\frac{60}{137}$	5	$-\frac{110}{137}$	51°
6	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$	$\frac{60}{147}$	6	$-\frac{20}{343}$	18°

exterior of closed regions in the right-hand plane. The angle subtended at the origin is $\alpha = 90°$ in the first two cases and $\alpha = 88°$ in the third case. The right part of Fig. 4 shows the upper halves of the regions of absolute stability of the 4-, 5-, and 6-step BDF's which include the negative real axis and make angles subtended at the origin of 73°, 51°, and 18°, respectively.

A short proof of the instability of the BDF formulas for $k \geq 7$ is found in [4]. BDF methods are used to solve stiff systems.

6.4 Numerical Differentiation Formulas

Numerical differentiation formulas (NDF) are a modification of BDF's. Letting

$$\nabla y_n = y_n - y_{n-1}$$

denote the backward difference of y_n, we rewrite the k-step BDF of order $p = k$ in the form

$$\sum_{m=1}^{k} \frac{1}{m} \nabla^m y_{n+1} = h f_{n+1}.$$

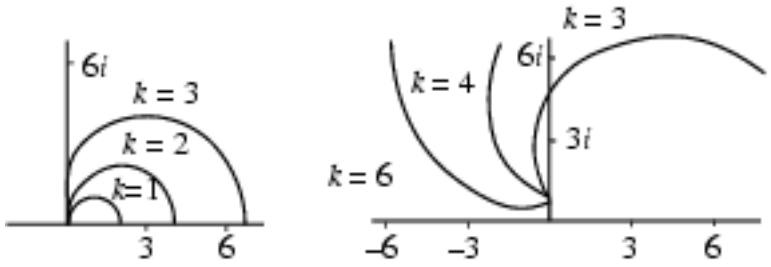

Figure 4: Left: Regions of absolute stability for k-step BDF for $k = 1, 2 \ldots, 6$. These regions include the negative real axis.

The algebraic equation for y_{n+1} is solved with a simplified Newton (chord) iteration. The iteration is started with the predicted value

$$y_{n+1}^{[0]} = \sum_{m=0}^{k} \frac{1}{m} \nabla^m y_n.$$

Then the k-step NDF of order $p = k$ is

$$\sum_{m=1}^{k} \frac{1}{m} \nabla^m y_{n+1} = h f_{n+1} + \kappa \gamma_k \left(y_{n+1} - y_{n+1}^{[0]} \right),$$

where κ is a scalar parameter and $\gamma_k = \sum_{j=1}^{k} 1/j$. The NDF's of order 1 to 5 are given in Table 5.

Table 5: Coefficients of the NDF methods.

k	κ	α_5	α_4	α_3	α_2	α_1	α_0	β_k	p	C_{p+1}	α
1	$-37/200$					1	-1	1	1	1	90°
2	$-1/9$				1	$-\frac{4}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	2	$-\frac{2}{9}$	90°
3	-0.0823			1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$	$\frac{6}{11}$	3	$-\frac{3}{22}$	80°
4	-0.0415		1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$	$\frac{12}{25}$	4	$-\frac{12}{125}$	66°
5	0	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	$\frac{60}{137}$	5	$-\frac{110}{137}$	51°

In [5], the choice of the number κ is a compromise made in balancing efficiency in step size and stability angle. Compared with the BDF's, there is a step ratio gain of 26% in NDF's of order 1, 2, and 3, 12% in NDF of order 4, and no change in NDF of order 5. The percent change in the stability angle is 0%, 0%, −7%, −10%, and 0%, respectively. No NDF of order 6 is considered because, in this case, the angle α is too small.

7 The Methods in the Matlab ODE Suite

The MATLAB ODE suite contains three explicit methods for nonstiff problems:

- The explicit Runge–Kutta pair `ode23` of orders 3 and 2,
- The explicit Runge–Kutta pair `ode45` of orders 5 and 4, of Dormand–Prince,
- The Adams–Bashforth–Moulton predictor-corrector pairs `ode113` of orders 1 to 13,

and two implicit methods for stiff systems:

- The implicit Runge–Kutta pair `ode23s` of orders 2 and 3,
- The implicit numerical differentiation formulas `ode15s` of orders 1 to 5.

All these methods have a built-in local error estimate to control the step size. Moreover `ode113` and `ode15s` are variable-order packages which use higher order methods and smaller step size when the solution varies rapidly.

The command `odeset` lets one create or alter the ode option structure.

The ODE suite is presented in a paper by Shampine and Reichelt [5] and the MATLAB `help` command supplies precise information on all aspects of their use. The codes themselves are found in the `toolbox/matlab/funfun` folder of MATLAB 5. For MATLAB 4.2 or later, it can be downloaded for free by `ftp` on `ftp.mathworks.com` in the `pub/mathworks/toolbox/matlab/funfun` directory. The second edition of the book by Ashino and Vaillancourt [6] on MATLAB 5 will contain a section on the `ode` methods.

In MATLAB 5, the command

`odedemo`

lets one solve 4 nonstiff problems and 15 stiff problems by any of the five methods in the suite. The two methods for stiff problems are also designed to solve nonstiff problems. The three nonstiff methods are poor at solving very stiff problems.

For graphing purposes, all five methods use interpolants to obtain, by default, four or, if specified by the user, more intermediate values of y between y_n and y_{n+1} to produce smooth solution curves.

7.1 The `ode23` method

The code `ode23` consists in a four-stage pair of embedded explicit Runge–Kutta methods of orders 2 and 3 with error control. It advances from y_n to y_{n+1} with the third-order method (so called local extrapolation) and controls the local error by taking the difference between the third-order and the second-order numerical solutions. The four stages are:

$$\begin{aligned} k1 &= hf(t_n, y_n), \\ k2 &= hf(t_n + (1/2)h, y_n + (1/2)k_1), \\ k3 &= hf(t_n + (3/4)h, y_n + (3/4)k_2), \\ k4 &= hf(t_n + h, y_n + (2/9)k_1 + (1/3)k_2 + (4/9)k_3), \end{aligned}$$

The first three stages produce the solution at the next time step:

$$y_{n+1} = y_n + (2/9)k_1 + (1/3)k_2 + (4/9)k_3,$$

and all four stages give the local error estimate:

$$E = -\frac{5}{72}\,k_1 + \frac{1}{12}\,k_2 + \frac{1}{9}\,k_2 - \frac{1}{8}\,k_4.$$

However, this is really a three-stage method since the first step at t_{n+1} is the same as the last step at t_n, that is $k_1^{[n+1]} = k_4^{[n]}$ (that is, a FSAL method).

The natural interpolant used in `ode23` is the two-point Hermite polynomial of degree 3 which interpolates y_n and $f(t_n, y_n)$ at $t = t_n$, and y_{n+1} and $f(t_{n+1}, t_{n+1})$ at $t = t_{n+1}$.

7.2 The `ode45` method

The code `ode45` is the Dormand-Prince pair DP5(4)7M with a high-quality "free" interpolant of order 4 that was communicated to Shampine and Reichelt [5] by Dormand and Prince. Since `ode45` can use long step size, the default is to use the interpolant to compute solution values at four points equally spaced within the span of each natural step.

7.3 The `ode113` method

The code `ode113` is a variable step variable order method which uses Adams–Bashforth–Moulton predictor-correctors of order 1 to 13. This is accomplish by monitoring the integration very closely. In the MATLAB graphics context, the monitoring is expensive. Although more than graphical accuracy is necessary for adequate resolution of moderately unstable problems, the high accuracy formulas available in `ode113` are not nearly as helpful in the present context as they are in general scientific computation.

7.4 The `ode23s` method

The code `ode23s` is a triple of modified implicit Rosenbrock methods of orders 3 and 2 with error control for stiff systems. It advances from y_n to y_{n+1} with the second-order method (that is, without local extrapolation) and controls the local error by taking the difference between the third- and second-order numerical solutions. Here is the algorithm:

$$\begin{aligned} f_0 &= hf(t_n, y_n), \\ k_1 &= W^{-1}(f_0 + hdT), \\ f_1 &= f(t_n + 0.5h, y_n + 0.5hk_1), \\ k_2 &= W^{-1}(f_1 - k_1) + k_1, \\ y_{n+1} &= y_n + hk_2, \\ f_2 &= f(t_{n+1}, y_{n+1}), \\ k3 &= W^{-1}[f_2 - c_{32}(k_2 - f_1) - 2(k_1 - f_0) + hdt], \\ \text{error} &\approx \frac{h}{6}\,(k_1 - 2k_2 + k_3), \end{aligned}$$

where

$$W = I - hdJ, \qquad d = 1/(2+\sqrt{2}\,), \qquad c_{32} = 6+\sqrt{2},$$

and

$$J \approx \frac{\partial f}{\partial y}(t_n, y_n), \qquad T \approx \frac{\partial f}{\partial t}(t_n, y_n).$$

This method is FSAL (First Step As Last). The interpolant used in `ode23s` is the quadratic polynomial in s:

$$y_{n+s} = y_n + h\left[\frac{s(1-s)}{1-2d}\,k_1 + \frac{s(s-2d)}{1-2d}\,k_2\right].$$

7.5 The `ode15s` method

The code `ode15s` for stiff systems is a quasi-constant step size implementation of the NDF's of order 1 to 5 in terms of backward differences. Backward differences are very suitable for implementing the NDF's in MATLAB because the basic algorithms can be coded compactly and efficiently and the way of changing step size is well-suited to the language. Options allow integration with the BDF's and integration with a maximum order less than the default 5. Equations of the form $M(t)y' = f(t,y)$ can be solved by the code `ode15s` for stiff problems with the `Mass` option set to `on`.

8 Solving Two Examples with Matlab

Our first example considers a non-stiff second-order ODE. Our second example considers the effect of a high stiffness ratio on the step size.

Example 1. Use the Runge–Kutta method of order 4 with fixed step size $h = 0.1$ to solve the second-order van der Pol equation

$$y'' + \big(y^2 - 1\big)y' + y = 0, \qquad y(0) = 0, \qquad y'(0) = 0.25, \tag{14}$$

on $0 \le x \le 20$, print every tenth value, and plot the numerical solution. Also, use the `ode23` code to solve (14) and plot the solution.

Solution. We first rewrite problem (14) as a system of two first-order differential equations by putting $y_1 = y$ and $y_2 = y_1'$,

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_2\big(1 - y_1^2\big) - y_1, \end{aligned}$$

with initial conditions $y_1(0) = 0$ and $y_2(0) = 0.25$.

Our MATLAB program will call the function M-file `exp1vdp.m`:

```
function yprime = exp1vdp(t,y); % Example 1.
yprime = [y(2); y(2).*(1-y(1).^2)-y(1)]; % van der Pol system
```

The following program applies the Runge–Kutta method of order 4 to the differential equation defined in the M-file `exp1vdp.m`:

```
clear
h = 0.1; t0= 0; tf= 21; % step size, initial and final times
y0 = [0 0.25]'; % initial conditions
n = ceil((tf-t0)/h); % number of steps

count = 2; print_control = 10; % when to write to output
t = t0; y = y0; % initialize t and y
output = [t0 y0']; % first row of matrix of printed values
w = [t0, y0']; % first row of matrix of plotted values
for i=1:n
 k1 = h*exp1vdp(t,y);            k2 = h*exp1vdp(t+h/2,y+k1/2);
 k3 = h*exp1vdp(t+h/2,y+k2/2);   k4 = h*exp1vdp(t+h,y+k3);
 z  = y + (1/6)*(k1+2*k2+2*k3+k4);
```

```
 t = t + h;
   if count > print_control
      output = [output; t z']; % augmenting matrix of printed values
      count = count - print_control;
   end
y = z;
w = [w; t z']; % augmenting matrix of plotted values
count = count + 1;
end
[output(1:11,:) output(12:22,:)] % print numerical values of solution
save w % save matrix to plot the solution
```

The command `output` prints the values of t, y_1, and y_2.

```
      t        y(1)     y(2)       t        y(1)     y(2)

      0          0    0.2500   11.0000   -1.9923   -0.2797
 1.0000     0.3586    0.4297   12.0000   -1.6042    0.7195
 2.0000     0.6876    0.1163   13.0000   -0.5411    1.6023
 3.0000     0.4313   -0.6844   14.0000    1.6998    1.6113
 4.0000    -0.7899   -1.6222   15.0000    1.8173   -0.5621
 5.0000    -1.6075    0.1456   16.0000    0.9940   -1.1654
 6.0000    -0.9759    1.0662   17.0000   -0.9519   -2.6628
 7.0000     0.8487    2.5830   18.0000   -1.9688    0.3238
 8.0000     1.9531   -0.2733   19.0000   -1.3332    0.9004
 9.0000     1.3357   -0.8931   20.0000    0.1068    2.2766
10.0000    -0.0939   -2.2615   21.0000    1.9949    0.2625
```

The following commands graph the solution.

```
load w % load values to produce the graph
subplot(2,2,1);  plot(w(:,1),w(:,2));  % plot RK4 solution
title('RK4 solution y_n for Example 1');  xlabel('t_n');  ylabel('y_n');
```

We now use the `ode23` code. The command

```
load w % load values to produce the graph
v = [0 21 -3 3 ]; % set t and y axes
subplot(2,2,1);
plot(w(:,1),w(:,2)); % plot RK4 solution
axis(v);
title('RK4 solution y_n for Example 1'); xlabel('t_n'); ylabel('y_n');
subplot(2,2,2);
[t,y] = ode23('exp1vdp',[0 21], y0);
plot(t,y(:,1)); % plot ode23 solution
axis(v);
title('ode23 solution y_n for Example 1'); xlabel('t_n'); ylabel('y_n');
```

The code `ode23` produces three vectors, namely `t` of (144 unequally-spaced) nodes and corresponding solution values `y(1)` and `y(2)`, respectively. The left and right parts of Fig. 5 show the plots of the solutions obtained by RK4 and `ode23`, respectively. It is seen that the two graphs are identical. □

In our second example we analyze the effect of the large stiffness ratio of a simple system of two differential equations with constant coefficients. Such problems are called pseudo-stiff since they are quite tractable by implicit methods.

Consider the initial value problem

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}' = \begin{bmatrix} 1 & 0 \\ 0 & 10^q \end{bmatrix} \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}, \qquad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{15}$$

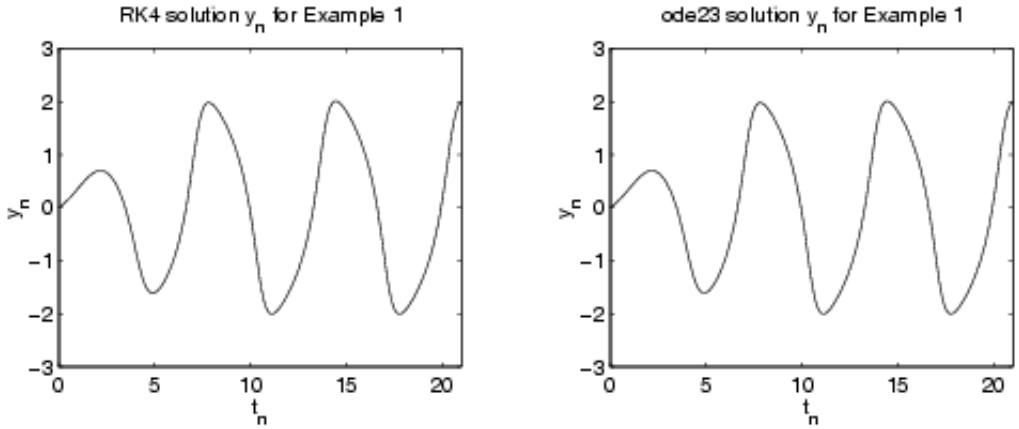

Figure 5: Graph of numerical solution of Example 1.

or

$$\boldsymbol{y}' = A\boldsymbol{y}, \qquad \boldsymbol{y}(0) = \boldsymbol{y}_0.$$

Since the eigenvalues of A are

$$\lambda_1 = 1, \qquad \lambda_2 = -10^q,$$

the stiffness ratio (11) of the system is

$$r = 10^q.$$

The solution is

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} e^{-t} \\ e^{-10^q t} \end{bmatrix}.$$

Even though the second part of the solution containing the fast decaying factor $\exp(-10^q t)$ for large q numerically disappears quickly, the large stiffness ratio continues to restrict the step size of any explicit schemes, including predictor-corrector schemes.

Example 2. Study the effect of the stiffness ratio on the number of steps used by the five **Matlab** ode codes in solving problem (15) with $q = 1$ and $q = 5$.

Solution. The function M-file `exp2.m` is

```
function uprime = exp2(t,u);  % Example 2
global q % global variable
A=[-1 0;0 -10^q]; % matrix A
uprime = A*u;
```

The following commands solve the non-stiff initial value problem with $q = 1$, and hence $r = e^{10}$, with relative and absolute tolerances equal to 10^{-12} and 10^{-14}, respectively. The option `stats on` requires that the code keeps track of the number of function evaluations.

```
clear;
global q; q=1;
tspan = [0 1]; y0 = [1 1]';
options = odeset('RelTol',1e-12,'AbsTol',1e-14,'Stats','on');
[x23,y23] = ode23('exp_camwa',tspan,y0,options);
[x45,y45] = ode45('exp_camwa',tspan,y0,options);
[x113,y113] = ode113('exp_camwa',tspan,y0,options);
[x23s,y23s] = ode23s('exp_camwa',tspan,y0,options);
[x15s,y15s] = ode15s('exp_camwa',tspan,y0,options);
```

Similarly, when $q = 5$, and hence $r = \exp(10^5)$, the program solves a pseudo-stiff initial value problem (15). Table 1 lists the number of steps used with $q = 1$ and $q = 5$ by each of the five methods of the ODE suite.

It is seen from the table that nonstiff solvers are hopelessly slow and very expensive in solving pseudo-stiff equations. □

Table 6: Number of steps used by each method with $q = 1$ and $q = 5$ with default relative and absolute tolerance $RT = 10^{-3}$ and $AT = 10^{-6}$ respectively, and same tolerance set at 10^{-12} and 10^{-14}, respectively.

(RT, AT)	$(10^{-3}, 10^{-6})$		$(10^{-12}, 10^{-14})$	
q	1	5	1	5
ode23	29	39 823	24 450	65 944
ode45	13	30 143	601	30 856
ode113	28	62 371	132	64 317
ode23s	37	57	30 500	36 925
ode15s	43	89	773	1 128

9 The `odeset` Options

Options for the five ode solvers can be listed by the `odeset` command (the default values are in curly brackets):

```
odeset
         AbsTol: [ positive scalar or vector {1e-6} ]
            BDF: [ on | {off} ]
         Events: [ on | {off} ]
    InitialStep: [ positive scalar ]
       Jacobian: [ on | {off} ]
      JConstant: [ on | {off} ]
       JPattern: [ on | {off} ]
           Mass: [ on | {off} ]
   MassConstant: [ on | off ]
       MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
        MaxStep: [ positive scalar ]
    NormControl: [ on | {off} ]
      OutputFcn: [ string ]
      OutputSel: [ vector of integers ]
         Refine: [ positive integer ]
         RelTol: [ positive scalar {1e-3} ]
          Stats: [ on | {off} ]
```

We first give a simple example of the use of ode options before listing the options in detail. The following commands solve the problem of Example 2 with different methods and different options.

```
[t, y]=ode23('exp2', [0 1], 0, odeset('RelTol', 1e-9, 'Refine', 6));
[t, y]=ode45('exp2', [0 1], 0, odeset(''AbsTol', 1e-12));
[t, y]=ode113('exp2', [0 1], 0, odeset('RelTol', 1e-9, 'AbsTol', 1e-12));
[t, y]=ode23s('exp2', [0 1], 0, odeset('RelTol', 1e-9, 'AbsTol', 1e-12));
[t, y]=ode15s('exp2', [0 1], 0, odeset('JConstant', 'on'));
```

The ode options are used in the demo problems in Sections 8 and 9 below. Others ways of inserting the options in the ode M-file are explained in [7].

The command ODESET creates or alters ODE OPTIONS structure as follows

- OPTIONS = ODESET('NAME1', VALUE1, 'NAME2', VALUE2, ...) creates an integrator options structure OPTIONS in which the named properties have the specified values. Any unspecified properties have default values. It is sufficient to type only the leading characters that uniquely identify the property. Case is ignored for property names.
- OPTIONS = ODESET(OLDOPTS, 'NAME1', VALUE1, ...) alters an existing options structure OLDOPTS.
- OPTIONS = ODESET(OLDOPTS, NEWOPTS) combines an existing options structure OLDOPTS with a new options structure NEWOPTS. Any new properties overwrite corresponding old properties.
- ODESET with no input arguments displays all property names and their possible values.

Here is the list of the `odeset` properties.

- `RelTol` : Relative error tolerance [positive scalar 1e-3] This scalar applies to all components of the solution vector and defaults to 1e-3 (0.1% accuracy) in all solvers. The estimated error in each integration step satisfies e(i) <= max(RelTol*abs(y(i)), AbsTol(i)).

- `AbsTol` : Absolute error tolerance [positive scalar or vector 1e-6] A scalar tolerance applies to all components of the solution vector. Elements of a vector of tolerances apply to corresponding components of the solution vector. AbsTol defaults to 1e-6 in all solvers.

- `Refine` : Output refinement factor [positive integer] This property increases the number of output points by the specified factor producing smoother output. Refine defaults to 1 in all solvers except ODE45, where it is 4. Refine does not apply if length(TSPAN) > 2.

- `OutputFcn` : Name of installable output function [string] This output function is called by the solver after each time step. When a solver is called with no output arguments, OutputFcn defaults to 'odeplot'. Otherwise, OutputFcn defaults to ''.

- `OutputSel` : Output selection indices [vector of integers] This vector of indices specifies which components of the solution vector are passed to the OutputFcn. OutputSel defaults to all components.

- `Stats` : Display computational cost statistics [on | {off}]

- `Jacobian` : Jacobian available from ODE file [on | {off}] Set this property 'on' if the ODE file is coded so that F(t, y, 'jacobian') returns dF/dy.

- `JConstant` : Constant Jacobian matrix dF/dy [on | {off}] Set this property 'on' if the Jacobian matrix dF/dy is constant.

- `JPattern` : Jacobian sparsity pattern available from ODE file [on | {off}] Set this property 'on' if the ODE file is coded so F([ ], [ ], 'jpattern') returns a sparse matrix with 1's showing nonzeros of dF/dy.

- `Vectorized` : Vectorized ODE file [on | {off}] Set this property 'on' if the ODE file is coded so that F(t, [y1 y2 ...]) returns [F(t, y1) F(t, y2) ...].

- `Events` : Locate events [on | off] Set this property 'on' if the ODE file is coded so that F(t, y, 'events') returns the values of the event functions. See ODEFILE.

- `Mass` : Mass matrix available from ODE file [on | {off}] Set this property 'on' if the ODE file is coded so that F(t, [ ], 'mass') returns time dependent mass matrix M(t).

- `MassConstan` : Constant mass matrix available from ODE file [on | {off}] Set this property 'on' if the ODE file is coded so that F(t, [ ], 'mass') returns a constant mass matrix M.

- `MaxStep` : Upper bound on step size [positive scalar] MaxStep defaults to one-tenth of the tspan interval in all solvers.

- `InitialStep` : Suggested initial step size [positive scalar] The solver will try this first. By default the solvers determine an initial step size automatically.

- `MaxOrder` : Maximum order of ODE15S [1 | 2 | 3 | 4 | {5}]

- `BDF` : Use Backward Differentiation Formulas in ODE15S [on | {off}] This property specifies whether the Backward Differentiation Formulas (Gear's methods) are to be used in ODE15S instead of the default Numerical Differentiation Formulas.

- `NormControl` : Control error relative to norm of solution [on | {off}] Set this property 'on' to request that the solvers control the error in each integration step with norm(e) <= max(RelTol*norm(y), AbsTol). By default the solvers use a more stringent component-wise error control.

10 Nonstiff Problems of the Matlab `odedemo`

10.1 The `orbitode` problem

ORBITODE is a restricted three-body problem. This is a standard test problem for non-stiff solvers stated in Shampine and Gordon, p. 246 ff in [8]. The first two solution components are coordinates of the body of infinitesimal mass, so plotting one against the other gives the orbit of the body around the other two bodies. The initial conditions have been chosen so as to make the orbit periodic. Moderately stringent tolerances are necessary to reproduce the qualitative behavior of the orbit. Suitable values are 1e-5 for RelTol and 1e-4 for AbsTol.

Because this function returns event function information, it can be used to test event location capabilities.

10.2 The `orbt2ode` problem

ORBT2ODE is the non-stiff problem D5 of Hull *et al.* [9] This is a two-body problem with an elliptical orbit of eccentricity 0.9. The first two solution components are coordinates of one body relative to the other body, so plotting one against the other gives the orbit. A plot of the first solution component as a function of time shows why this problem needs a small step size near the points of closest approach. Moderately stringent tolerances are necessary to reproduce the qualitative behavior of the orbit. Suitable values are 1e-5 for RelTol and 1e-5 for AbsTol. See [10], p. 121.

10.3 The `rigidode` problem

RIGIDODE solves Euler's equations of a rigid body without external forces.

This is a standard test problem for non-stiff solvers proposed by Krogh. The analytical solutions are Jacobi elliptic functions accessible in MATLAB. The interval of integration $[t_0, t_f]$ is about 1.5 periods; it is that for which solutions are plotted on p. 243 of Shampine and Gordon [8].

RIGIDODE([ ], [ ], 'init') returns the default TSPAN, Y0, and OPTIONS values for this problem. These values are retrieved by an ODE Suite solver if the solver is invoked with empty TSPAN or Y0 arguments. This example does not set any OPTIONS, so the third output argument is set to empty [ ] instead of an OPTIONS structure created with ODESET.

10.4 The `vdpode` problem

VDPODE is a parameterizable van der Pol equation (stiff for large mu). VDPODE(T, Y) or VDPODE(T, Y, [ ], MU) returns the derivatives vector for the van der Pol equation. By default, MU is 1, and the problem is not stiff. Optionally, pass in the MU parameter as an additional parameter to an ODE Suite solver. The problem becomes stiffer as MU is increased.

For the stiff problem, see Subsection 11.15.

11 Stiff Problems of the Matlab `odedemo`

11.1 The `a2ode` and `a3ode` problems

A2ODE and A3ODE are stiff linear problems with real eigenvalues (problem A2 of [11]). These nine- and four-equation systems from circuit theory have a constant tridiagonal Jacobian and also a constant partial derivative with respect to t because they are autonomous.

Remark 1. When the ODE solver JConstant property is set to 'off', these examples test the effectiveness of schemes for recognizing when Jacobians need to be refreshed. Because the Jacobians are constant, the ODE solver property JConstant can be set to 'on' to prevent the solvers from unnecessarily recomputing the Jacobian, making the integration more reliable and faster.

11.2 The `b5ode` problem

B5ODE is a stiff problem, linear with complex eigenvalues (problem B5 of [11]). See Ex. 5, p. 298 of Shampine [10] for a discussion of the stability of the BDFs applied to this problem and the role of the maximum order permitted (the MaxOrder property accepted by ODE15S). ODE15S solves this problem efficiently if the maximum order of the NDFs is restricted to 2.

This six-equation system has a constant Jacobian and also a constant partial derivative with respect to t because it is autonomous. Remark 1 applies to this example.

11.3 The `buiode` problem

BUIODE is a stiff problem with analytical solution due to Bui. The parameter values here correspond to the stiffest case of [12]; the solution is

$$y(1) = e^{-4t}, \qquad y(2) = e^{-t}.$$

11.4 The `brussode` problem

BRUSSODE is a stiff problem modelling a chemical reaction (the Brusselator) [1]. The command `BRUSSODE(T, Y)` or `BRUSSODE(T, Y, [ ], N)` returns the derivatives vector for the Brusselator problem. The parameter $N >= 2$ is used to specify the number of grid points; the resulting system consists of $2N$ equations. By default, N is 2. The problem becomes increasingly stiff and increasingly sparse as N is increased. The Jacobian for this problem is a sparse matrix (banded with bandwidth 5).

`BRUSSODE([ ], [ ], 'jpattern')` or `BRUSSODE([ ], [ ], 'jpattern', N)` returns a sparse matrix of 1's and 0's showing the locations of nonzeros in the Jacobian $\partial F/\partial Y$. By default, the stiff solvers of the ODE Suite generate Jacobians numerically as full matrices. However, if the ODE solver property `JPattern` is set to 'on' with ODESET, a solver calls the ODE file with the flag 'jpattern'. The ODE file returns a sparsity pattern that the solver uses to generate the Jacobian numerically as a sparse matrix. Providing a sparsity pattern can significantly reduce the number of function evaluations required to generate the Jacobian and can accelerate integration. For the BRUSSODE problem, only 4 evaluations of the function are needed to compute the $2N \times 2N$ Jacobian matrix.

11.5 The `chm6ode` problem

CHM6ODE is the stiff problem CHM6 from Enright and Hull [13]. This four-equation system models catalytic fluidized bed dynamics. A small absolute error tolerance is necessary because y(:,2) ranges from 7e-10 down to 1e-12. A suitable AbsTol is 1e-13 for all solution components. With this choice, the solution curves computed with `ode15s` are plausible. Because the step sizes span 15 orders of magnitude, a loglog plot is appropriate.

11.6 The `chm7ode` problem

CHM7ODE is the stiff problem CHM7 from [13]. This two-equation system models thermal decomposition in ozone.

11.7 The `chm9ode` problem

CHM9ODE is the stiff problem CHM9 from [13]. It is a scaled version of the famous Belousov oscillating chemical system. There is a discussion of this problem and plots of the solution starting on p. 49 of Aiken [14]. Aiken provides a plot for the interval $[0, 5]$, an interval of rapid change in the solution. The default time interval specified here includes two full periods and part of the next to show three periods of rapid change.

11.8 The `d1ode` problem

D1ODE is a stiff problem, nonlinear with real eigenvalues (problem D1 of [11]). This is a two-equation model from nuclear reactor theory. In [11] the problem is converted to autonomous form, but here it is solved in its original non-autonomous form. On page 151 in [15], van der Houwen provides the reference solution values

$$t = 400, \qquad y(1) = 22.24222011, \qquad y(2) = 27.11071335$$

11.9 The `fem1ode` problem

FEM1ODE is a stiff problem with a time-dependent mass matrix,

$$M(t)y' = f(t, y).$$

Remark 2. FEM1ODE(T, Y) or FEM1ODE(T, Y, [ ], N) returns the derivatives vector for a finite element discretization of a partial differential equation. The parameter N controls the discretization, and the resulting system consists of N equations. By default, N is 9.

FEM1ODE(T, [ ], 'mass') or FEM1ODE(T, [ ], 'mass', N) returns the time-dependent mass matrix M evaluated at time T. By default, ODE15S solves systems of the form

$$y' = f(t, y).$$

However, if the ODE solver property Mass is set to 'on' with ODESET, the solver calls the ODE file with the flag 'mass'. The ODE file returns a mass matrix that the solver uses to solve

$$M(t)y' = f(t, y).$$

If the mass matrix is a constant M, then the problem can be also be solved with ODE23S.

FEM1ODE also responds to the flag 'init' (see RIGIDODE).

For example, to solve a 20×20 system, use

[t, y] = ode15s('fem1ode', [ ], [ ], [ ], 20);

11.10 The `fem2ode` problem

FEM2ODE is a stiff problem with a time-independent mass matrix,

$$My' = f(t, y).$$

Remark 2 applies to this example, which can also be solved by `ode23s` with the command

[T, Y] = ode23s('fem2ode', [ ], [ ], [ ], 20).

11.11 The `gearode` problem

GEARODE is a simple stiff problem due to Gear as quoted by van der Houwen [15] who, on page 148, provides the reference solutionvalues

$$t = 50, \qquad y(1) = 0.5976546988, \qquad y(2) = 1.40234334075$$

11.12 The `hb1ode` problem

HB1ODE is the stiff problem 1 of Hindmarsh and Byrne [16]. This is the original Robertson chemical reaction problem on a very long interval. Because the components tend to a constant limit, it tests reuse of Jacobians. The equations themselves can be unstable for negative solution components, which is admitted by the error control. Many codes can, therefore, go unstable on a long time interval because a solution component goes to zero and a negative approximation is entirely possible. The default interval is the longest for which the Hindmarsh and Byrne code EPISODE is stable. The system satisfies a conservation law which can be monitored:

$$y(1) + y(2) + y(3) = 1.$$

11.13 The `hb2ode` problem

HB2ODE is the stiff problem 2 of [16]. This is a non-autonomous diurnal kinetics problem that strains the step size selection scheme. It is an example for which quite small values of the absolute error tolerance are appropriate. It is also reasonable to impose a maximum step size so as to recognize the scale of the problem. Suitable values are an AbsTol of 1e-20 and a MaxStep of 3600 (one hour). The time interval is 1/3; this interval is used by Kahaner, Moler, and Nash, p. 312 in [17], who display the solution on p. 313. That graph is a semilog plot using solution values only as small as 1e-3. A small threshold of 1e-20 specified by the absolute error control tests whether the solver will keep the size of the solution this small during the night time. Hindmarsh and Byrne observe that their variable order code resorts to high orders during the day (as high as 5), so it is not surprising that relatively low order codes like ODE23S might be comparatively inefficient.

11.14 The `hb3ode` problem

HB3ODE is the stiff problem 3 of Hindmarsh and Byrne [16]. This is the Hindmarsh and Byrne mockup of the diurnal variation problem. It is not nearly as realistic as HB2ODE and is quite special in that the Jacobian is constant, but it is interesting because the solution exhibits quasi-discontinuities. It is posed here in its original non-autonomous form. As with HB2ODE, it is reasonable to impose a maximum step size so as to recognize the scale of the problem.

A suitable value is a MaxStep of 3600 (one hour). Because y(:,1) ranges from about 1e-27 to about 1.1e-26, a suitable AbsTol is 1e-29.

Because of the constant Jacobian, the ODE solver property JConstant prevents the solvers from recomputing the Jacobian, making the integration more reliable and faster.

11.15 The `vdpode` problem

VDPODE is a parameterizable van der Pol equation (stiff for large mu) [18]. VDPODE(T, Y) or VDPODE(T, Y, [ ], MU) returns the derivatives vector for the van der Pol equation. By default, MU is 1, and the problem is not stiff. Optionally, pass in the MU parameter as an additional parameter to an ODE Suite solver. The problem becomes more stiff as MU is increased.

When MU is 1000 the equation is in relaxation oscillation, and the problem becomes very stiff. The limit cycle has portions where the solution components change slowly and the problem is quite stiff, alternating with regions of very sharp change where it is not stiff (quasi-discontinuities). The initial conditions are close to an area of slow change so as to test schemes for the selection of the initial step size.

VDPODE(T, Y, 'jacobian') or VDPODE(T, Y, 'jacobian', MU) returns the Jacobian matrix $\partial F/\partial Y$ evaluated analytically at (T, Y). By default, the stiff solvers of the ODE Suite approximate Jacobian matrices numerically. However, if the ODE Solver property Jacobian is set to 'on' with ODESET, a solver calls the ODE file with the flag 'jacobian' to obtain $\partial F/\partial Y$. Providing the solvers with an analytic Jacobian is not necessary, but it can improve the reliability and efficiency of integration.

VDPODE([ ], [ ], 'init') returns the default TSPAN, Y0, and OPTIONS values for this problem (see RIGIDODE). The ODE solver property Vectorized is set to 'on' with ODESET because VDPODE is coded so that calling VDPODE(T, [Y1 Y2 ...]) returns [VDPODE(T, Y1) VDPODE(T, Y2) ...] for scalar time T and vectors Y1, Y2,... The stiff solvers of the ODE Suite take advantage of this feature when approximating the columns of the Jacobian numerically.

12 Concluding Remarks

Ongoing research in explicit and implicit Runge–Kutta pairs, and hybrid methods, which incorporate function evaluations at off-step points in order to lower the stepnumber of a linear multistep method without reducing its order (see [19], [20], [21]), may, in the future, improve the MATLAB ODE suite.

Acknowledgments

The authors thank the referee for deep and constructive remarks which improved the paper considerably. This paper is an expanded version of a lecture given by the third author at Ritsumeikan University, Kusatsu, Shiga, 525-8577 Japan, University upon the kind invitation of Professor Osanobu Yamada whom the authors thank very warmly.

References

[1] E. Hairer and G. Wanner, *Solving ordinary differential equations II, stiff and differential-algebraic problems*, Springer-Verlag, Berlin, 1991, pp. 5–8.

[2] J. D. Lambert, *Numerical methods for ordinary differential equations. The initial value problem*, Wiley, Chichester, 1991.

[3] J. R. Dormand and P. J. Prince, *A family of embedded Runge–Kutta formulae*, J. Computational and Applied Mathematics, **6**(2) (1980), 19–26.

[4] E. Hairer and G. Wanner, *On the instability of the BDF formulas*, SIAM J. Numer. Anal., **20**(6) (1983), 1206–1209.

[5] L. F. Shampine and M. W. Reichelt, *The Matlab ODE suite*, SIAM J. Sci. Comput., **18**(1), (1997) 1–22.

[6] R. Ashino and R. Vaillancourt, *Hayawakari Matlab (Introduction to Matlab)*, Kyoritsu Shuppan, Tokyo, 1997, xvi–211 pp., 6th printing, 1999 (in Japanese). (Korean translation, 1998.)

[7] *Using MATLAB*, Version, 5.1, The MathWorks, Chapter 8, Natick, MA, 1997.

[8] L. F. Shampine and M. K. Gordon, *Computer solution of ordinary differential equations*, W.H. Freeman & Co., San Francisco, 1975.

[9] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., **9**(4) (1972) 603–637.

[10] L. F. Shampine, *Numerical solution of ordinary differential equations*, Chapman & Hall, New York, 1994.

[11] W. H. Enright, T. E. Hull, and B. Lindberg, *Comparing numerical methods for stiff systems of ODEs*, BIT **15**(1) (1975), 10–48.

[12] L. F. Shampine, *Measuring stiffness*, Appl. Numer. Math., **1**(2) (1985), 107–119.

[13] W. H. Enright and T. E. Hull, *Comparing numerical methods for the solution of stiff systems of ODEs arising in chemistry*, in Numerical Methods for Differential Systems, L. Lapidus and W. E. Schiesser eds., Academic Press, Orlando, FL, 1976, pp. 45–67.

[14] R. C. Aiken, ed., *Stiff computation*, Oxford Univ. Press, Oxford, 1985.

[15] P. J. van der Houwen, *Construction of integration formulas for initial value problems*, North-Holland Publishing Co., Amsterdam, 1977.

[16] A. C. Hindmarsh and G. D. Byrne, *Applications of EPISODE: An experimental package for the integration of ordinary differential equations*, in Numerical Methods for Differential Systems, L. Lapidus and W. E. Schiesser eds., Academic Press, Orlando, FL, 1976, pp. 147–166.

[17] D. Kahaner, C. Moler, and S. Nash, *Numerical methods and software*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[18] L. F. Shampine, *Evaluation of a test set for stiff ODE solvers*, ACM Trans. Math. Soft., **7**(4) (1981) 409–420.

[19] J. D. Lambert, *Computational methods in ordinary differential equations*, Wiley, London, 1973, Chapter 5.

[20] J. C. Butcher, *The numerical analysis of ordinary differential equations. Runge–Kutta and general linear methods*, Wiley, Chichester, 1987, Chapter 4.

[21] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations I, nonstiff problems*, Springer-Verlag, Berlin, 1987, Section III.8.