Numerical methods for local models

T.I. Lakoba*

Department of Mathematics and Statistics, 16 Colchester Ave.,
University of Vermont, Burlington, VT 05401, USA

February 14, 2007

*lakobati@cems.uvm.edu, 1 (802) 656-2610

Article synopsis

Numerical methods commonly used for solving ordinary differential equations, such as Euler and trapezoidal (both explicit and implicit), Runge–Kutta, an multistep ones are presented and compared with each other in terms of their accuracy and stability. The stability of any given method is shown to be a critical factor determining whether the method is useful. It is emphasized that numerical methods for conservative and nonconservative models must possess different stability features, and it is further shown that Runge–Kutta methods possess both of these features. The idea of methods with an adaptive step size is described. The phenomenon of numerical stiffness is explained. Suitable built-in commands of Matlab are mentioned, and an example of their usage is given.

Keywords: Numerical solution of differential equations; local models.

EDITORS, please include proper keywords or send me the list of them to pick. I simply don't know what else to put here.

Elementary methods for a single ODE

Ordinary differential equations (ODEs) are often found to model ecological systems that describe the evolution in time. ODEs arise when the system satisfies two requirements. First, the evolution is to be sufficiently "smooth". That is, the change of the system's state from one observation moment to the next should be relatively small. These small changes may accumulate into drastic ones given a sufficiently long evolution time. Second, one must be able to view the system as behaving as a whole, undivisible entity. For example, the system should be assumed to have no spatial extent, or, if it does have spatial dimensions, then the state of the system must be the same in all of its spatial locations. In other words, the system must be uniform, or local. If the system does not meet this requirement, it is referred to as distributed. Methods for, and examples of, such distributed systems are found in the companion article by the same author in this volume.

Some of the more simple ODEs can be solved analytically. For example, the classical Lotka-Volterra model describing dynamics of a predator-prey system was formulated as a system of two ODEs and studied analytically. However, as equations become more complex, bearing nonlinearities, or as the number of equations becomes large, analytical solution becomes impossible. In that case ODEs have to be treated numerically.

This article surveys numerical methods for solving evolution problems described by an ODE

$$\dot{u}(t) = f(u,t), \qquad u(t_0) = u_0 \tag{1}$$

or systems of such equations; here $\dot{u} \equiv du/dt$. Let $t_n = t_0 + n\tau$, where $n = 1, 2, \ldots$ and τ is a (small) time step. Denote $u_n \equiv u(t_n)$ and U_n to be, respectively, the exact and the numerical solutions at time t_n. Replacing $\dot{u}$ with $(U_{n+1} - U_n)/\tau$ and denoting $f_n \equiv f(U_n, t_n)$, one obtains the explicit Euler method:

$$U_{n+1} = U_n + \tau\, f_n\,. \tag{2}$$

To determine how accurately this approximates (1), one compares the l.h.s. of (2) with u_{n+1}:

$$u_{n+1} = u(t_n + \tau) = u_n + \tau\, \dot{u}_n + O(\tau^2) = u_n + \tau\, f_n + O(\tau^2)\,. \tag{3}$$

The above notation $O(\tau^m)$, which will also be used throughout this article, stands for any quantity that vanishes at the same rate as τ^m as τ tends to zero. For example, both $-0.1\tau^2 + 15\tau^3$ and $\tau^2/(1+\tau\cos(2-\tau))$ are $O(\tau^2)$. Note also that all of $O(\tau^m)+O(\tau^m)$, $O(\tau^m)-O(\tau^m)$, and $O(\tau^m)+O(\tau^{m+1})$ are $O(\tau^m)$. In the second equality in (3), one used the first two terms of the Taylor expansion:

$$u(t+\tau) = u(t) + \dot{u}(t)\tau + \frac{1}{2!}\ddot{u}(t)\tau^2 + O(\tau^3)\,, \tag{4}$$

and in the third, Eq. (1) for $\dot{u}$.

In the accuracy analysis, one conventionally assumes that at steps before the last one, the exact and numerical solutions coincide, i.e., $u_n = U_n$. Then (2) and (3) yield the local error (the error made at each computational step):

$$U_n - u_n = O(\tau^2). \tag{5}$$

The errors at individual steps accumulate into the global error at the end of the computational interval. There are $(t_{\text{final}} - t_0)/\tau = O(1/\tau)$ such steps, whence

$$\text{global error} \;=\; \frac{\text{const}}{\tau} \cdot \text{local error}\,. \tag{6}$$

In particular, the global error of the explicit Euler method is $O(\tau)$. Methods whose global error is $O(\tau^m)$ are called mth-order methods. When $m > 0$, the global error tends to zero with τ, and the corresponding methods are called consistent.

The global error can be small only if the constant in (6) does not grow with n. This motivates the notion of stability of a numerical method. A method is stable if its solution, which is supposed to be close to the exact solution at some time $\bar{t}$, remains close to the exact solution at all later times. Thus, for a numerical method to provide a useful approximation, i.e., converge, to the exact solution (when τ is sufficiently small), the method must be both consistent and stable. This is usually stated as the Lax Theorem:

$$\text{Consistency} + \text{Stability} \;\Rightarrow\; \text{Convergence}. \tag{7}$$

Let $u + \delta u$ be a solution that is close to some given solution u at $\bar{t}$; i.e., δu is much smaller than u. This new solution satisfies (1) with u being replaced by $u + \delta u$. Subtracting these two equations and neglecting terms of order $O((\delta u)^2)$, one obtains:

$$\dot{\delta u} \;=\; [f_u(u(\bar{t}), \bar{t})]\; \delta u\,, \tag{8}$$

where $f_u \equiv \partial_u f(u,t)$. The quantity in brackets in (8) is a fixed number (rather than a function of t) for any given value $\bar{t}$, and it is this number that determined whether the initially small deviation δu between the two solutions grows or decays, i.e., the stability of the solution. Therefore, the stability of numerical methods for (1) is tested on an equation that has the form of (8) but, by convention, is written for u rather than for δu:

$$\dot{u} = \lambda\, u, \qquad \lambda = \text{const}\,. \tag{9}$$

For reasons that will be explained when we consider systems of ODEs, λ is allowed to have complex values, but with $\mathrm{Re}(\lambda) \le 0$. (If $\mathrm{Re}(\lambda) > 0$, the solution of (8) grows as $\exp[\lambda(t - t_0)]$, and hence it does not make sense to require that the numerical solution stay close to the exact one if the latter itself is unstable.) Applying the explicit Euler method (2) to (9) yields:

$$U_{n+1} = (1 + \lambda\tau)U_n, \tag{10}$$

which means that this method is stable when

$$|1 + \lambda\tau| \le 1 \tag{11}$$

When λ (< 0) is real, (11) implies the restriction on the step size of the form: $\tau < 2/(-\lambda)$. In general, when $\lambda \equiv \lambda_r + i\lambda_i$ is complex, (11) is rewritten as $(1 + \lambda_r\tau)^2 + (\lambda_i\tau)^2 \le 1$, which defines the inside of the circle shown in Fig. 1 by the solid line. < EDITORS: Fig. 1 near here. >

If in (1), one replaces $\dot{u}$ with $(U_n - U_{n-1})/\tau$, the resulting method becomes the implicit Euler method:

$$U_{n+1} = U_n + \tau\, f_{n+1}\,. \tag{12}$$

Folowing the analysis above, one can show that its global error is also $O(\tau)$. The disadvantage of this method compared to (2) is that to implement (12), one has to solve a nonlinear equation for U_{n+1} at each step. The advantage, however, is that this method has a much greater stability region than method (2): similarly to the above, one shows that the implicit Euler method is stable for

$$|1 - \lambda\tau| \geq 1\,, \tag{13}$$

which defines the outside of the circle shown in Fig. 1 by the dashed line.

The accuracy of the Euler methods can be improved using the following observation. If f in (1) were constant, methods (2) and (12) would have been exact, because in this case, the slope of the evolution of $u(t)$, determined by the f, would have been constant. Thus, the error in, e.g., (5) occurs because the actual slope $f(u,t)$ changes between t_n and t_{n+1}. Therefore, to improve the accuracy of the method, one can take the average of the slopes at t_n and t_{n+1}. The result is the modified implicit Euler method (also called implicit trapezoidal or implicit Heun method):

$$U_{n+1} = U_n + \frac{\tau}{2}\,(f_n + f_{n+1})\,. \tag{14}$$

One can show that the global error of this method is $O(\tau^2)$ and its stability region is the entire left-half plane (i.e. $\lambda_r\tau \leq 0$). Thus, this consistent method is stable for any step size τ, and hence converges (see (7)) to the exact solution of (1) for any value of τ. However, the disadvantage of this method is the same as that of (12): in general, one has to solve a nonlinear equation to obtain U_{n+1}.

To avoid solving nonlinear equations, one can replace U_{n+1} on the r.h.s. of (14) with its estimate obtained from (2). This yields the modified explicit Euler, or Heun or trapezoidal, method:

$$\bar{U} = U_n + \tau\, f_n\,, \tag{15}$$

$$U_{n+1} = U_n + \frac{\tau}{2}\left(f_n + f(\bar{U}, t_{n+1})\right)\,. \tag{16}$$

Its global error is also $O(\tau^2)$ and the stability condition is

$$\left|1 + \lambda\tau + \frac{1}{2}(\lambda\tau)^2\right|^2 \leq 1\,; \tag{17}$$

the corresponding region is the inside of the oval (thin solid line) in Fig. 2. Notice that the expression on the l.h.s. of (17) is just the $O(\tau^2)$-accurate Taylor expansion (4) of $e^{\lambda\tau}$, which correlates with the Heun method having the global error of $O(\tau^2)$. When λ is a real number, the stability condition of this method is $\tau < 2/(-\lambda)$, i.e. the same as that for the explicit Euler method (2).

A family of methods, called Runge–Kutta (RK) methods, generalizes the explicit Heun method and allows one to construct methods with higher accuracy. This family of methods has

the form:

$$
\begin{aligned}
&U_{n+1} = U_n + (c_1 R_1 + c_2 R_2 + c_3 R_3 + \dots);\\
&R_1 = \tau f_n\,,\\
&R_2 = \tau f(U_n + \beta_{21} R_1,\, t_n + \alpha_2 \tau),\\
&R_3 = \tau f(U_n + \beta_{31} R_1 + \beta_{32} R_2,\, t_n + \alpha_3 \tau),\\
&\text{etc.}
\end{aligned}
\tag{18}
$$

where the c's, α's, and β's are some properly chosen numeric coefficients. (Method (15), (16) results for $c_2 = c_3 = \dots = 0$.) Historically, the most popular RK method is the following 4th-order method:

$$
\begin{aligned}
&U_{n+1} = U_n + \frac{1}{6}\,(R_1 + 2R_2 + 2R_3 + R_4);\\
&R_1 = \tau f_n\,,\\
&R_2 = \tau f\left(U_n + \frac{1}{2} R_1,\, t_n + \frac{1}{2}\tau\right),\\
&R_3 = \tau f\left(U_n + \frac{1}{2} R_2,\, t_n + \frac{1}{2}\tau\right),\\
&R_4 = \tau f\,(U_n + R_3,\, t_n + \tau)\,.
\end{aligned}
\tag{19}
$$

The stability region of this method is the inside of the contour shown with the thick solid line in Fig. 2. When λ is a real number, the stability condition of this method is $\tau < 2.79/(-\lambda)$. Thus, the advantage of the RK method (19) over the explicit Euler and Heun methods is not only its higher accuracy but also the more relaxed stability condition on the step size τ. Yet another advantage of this method will be pointed out later on. < EDITORS: Fig. 2 near here. >

Adaptive methods

Adaptive methods are currently the default methods for solving ODEs in major computing software. These methods, which can adapt the step size to the conditions of the problem, are most useful when the coefficients in the problem change very rapidly over some time intervals and smoothly otherwise. One simple example here is the motion of a skydiver: the air resistance changes abruptly at the moment when the parachute opens. The coefficients in an equation, say (1), determine the value of $\partial_u f$, which is (see (9)) the prototype of the parameter λ in the test equation (10). Thus, a drastic change in $\partial_u f$ will require a corresponding change in τ to preserve both the accuracy and the stability of the numerical method. On the other hand, one obviously wants to take as large a time step as possible to minimize the computational time.

Let us emphasize that in adaptive methods, one controls the local error, and not the global error, of the solution. Indeed, the only way to control the global error is to run the simulations more than once. For example, one can run a simulation with the step τ and then repeat it with the step $\tau/2$ to verify that the difference between the two solutions is within a prescribed accuracy. Although this can be done occasionally, it is computationally inefficient to do so routinely within the code. Therefore, error control algorithms in adaptive methods ensure that the local error at each step is less than a given threshold, ε_{loc}. Next, let us assume for the moment that the exact solution $u(t)$ is known. Then conceptually, the steps of an error control

algorithm are the following. At each t_n, compute the local error $\epsilon_n = |U_n - u_n|$. If $\epsilon_n < \varepsilon_{\rm loc}$, accept the solution, multiply the next step size by $\kappa(\varepsilon_{\rm loc}/\epsilon_n)^{1/(m+1)}$ (where $\kappa \sim 0.8$ is a "safety" factor and m is the order of the method), and proceed to the next step. If $\epsilon_n > \varepsilon_{\rm loc}$, then multiply the step size by $\kappa(\varepsilon_{\rm loc}/\epsilon_n)^{1/(m+1)} < 1$, re-calculate the solution at this step, and check the new local error. If this error is acceptable, proceed to the next step. If not, repeat this step again. Now in reality, the exact solution u_n is not known. Then one can use, along with the given mth-order numerical method, another method of a higher order whose (more accurate) solution would play the role of the exact solution u_n above. To make this idea work time-efficiently, the more accurate method should share some of the computational steps with the original one, as first proposed by in 1970 by E. Fehlberg. He found a pair of the RK methods (18) where six coefficients $R_1 \ldots R_6$ are computed to obtain the 4th- and 5th-order accurate solutions, $U_n^{[4]}$ and $U_n^{[5]}$. Then the local error is computed as $\epsilon_n = |U_n^{[5]} - U_n^{[4]}|$. One now has a choice which of the two solutions one should accept as the output U_n of the numerical method, and the common sense suggests setting $U_n = U_n^{[5]}$. Thus, this adaptive method computes a 5th-order accurate solution $U_n^{[5]}$ while controlling the error of the less accurate 4th-order solution $U_n^{[4]}$. Other adaptive methods operate similarly. Such methods are commonly referred to as RK-Fehlberg, or embedded RK, methods.

A method from this family of methods proposed by J. Dormand and P. Prince is used in the Matlab's built-in command `ode45`, which computes a 5th-order solution of a given system of ODEs using an adaptive step size. < EDITORS: Please don't change the typesetter fornt for Matlab commands to the regular font. Using the typesetter font in this case is *not* equivalent to italicizing, boldfacing, or underlying a term. Rather, it is analogous to italicizing latin names, as shown in the example articles you posted for the information of the authors. The use of such font for Matlab commands is accepted as a convention in the literature on computational methods. Thank you for not changing the font. > For an example of a code using a similar command see Fig. 5 below. Analogous built-in adaptive integration commands exist in Fortran and C.

Multistep methods

These methods use the numerical solution at t_n and also at earlier times, t_{n-1}, t_{n-2}, etc., to obtain the solution at t_{n+1}. (In contrast, methods like (19) use the solution only at the time t_n to obtain the solution at t_{n+1} and hence are called single-step methods.) The idea behind multistep methods is to use the solution computed at those earlier steps to predict not only the slope, given by the r.h.s. of (1), of the solution at t_n, but also the curvature (the second derivative) and possibly higher-order derivatives of the solution. This allows one to approximate it at t_{n+1} with an accuracy higher than that achieved by (2). For example, the formula for a second-order-accurate, two-step method for (1) can be derived from the Taylor expansion of the same order (see (4)):

$$U_{n+1} = U_n + \tau U_n' + \frac{\tau^2}{2} U_n'' = U_n + \tau f_n + \frac{\tau}{2}(f_n - f_{n-1})\,. \tag{20}$$

In deriving (20), one uses $U_n' = f_n$ and its corollary: $U_n'' = (f_n - f_{n-1})/\tau + O(\tau)$, and omits terms of order $O(\tau^3)$ and higher. To start this method, one uses f_0 from the initial condition

and f_1 found by some single-step method. Another well-known two-step second-order method is a so called leap-frog method:

$$U_{n+1} = U_{n-1} + \tau f_n \,. \tag{21}$$

(It should be noted that this method is unstable for the test equation (10) for any $\mathrm{Re}\lambda < 0$; its stability region is the segment along the imaginary λ-axis shown in Fig. 1.) Formulae for higher-order multistep methods, known as Adams methods, can be found in most textbooks. The advantage of multistep methods over the single-step RK methods is that the latter require at least m function evaluations per step for an mth-order-accurate RK method (e.g., (19) requires 4 function evaluations), while a multistep method can achieve the same accuracy with only one function evaluation for any order m. Thus, multistep methods are faster than the RK ones. The main disadvantage of the multistep methods is that it is difficult to make them use adaptive step size, because their formulae are inherently based on the assumption that all steps have the same size τ. Another disadvantage of multistep methods is that they have smaller stability regions, which shrink with increasing the method's order. Therefore, currently these methods are not widely used in commercial software, where the adaptive embedded RK methods are used instead.

Methods for systems of ODEs

The extension of all of the above methods to higher-order equations or to K coupled equations,

$$\dot{u}^{(k)} = f^{(k)}(\vec{u}, t), \qquad k = 1, \ldots K, \quad \vec{u} = (u^{(1)}, \ldots u^{(K)}), \tag{22}$$

is straightforward. Note that any higher-order ODE can be put in the form (22). For example, $\dddot{u} = f(u, \dot{u}, \ddot{u}, t)$ is written in this form as follows: $u^{(1)} \equiv u$, $\dot{u}^{(1)} = u^{(2)}$, $\dot{u}^{(2)} = u^{(3)}$, $\dot{u}^{(3)} = f(u^{(1)}, u^{(2)}, u^{(3)}, t)$. The extension of, e.g., the RK method (19) to (22) is, for $k = 1, \ldots, K$:

$$\begin{aligned}
U^{(k)}_{n+1} &= U^{(k)}_n + \frac{1}{6}\,(R^{(k)}_1 + 2R^{(k)}_2 + 2R^{(k)}_3 + R^{(k)}_4); \\
R^{(k)}_1 &= \tau f^{(k)}_n \,, \\
R^{(k)}_2 &= \tau f^{(k)} \left(\vec{U}_n + \frac{1}{2}\vec{R}_1,\, t_n + \frac{1}{2}\tau\right), \\
R^{(k)}_3 &= \tau f^{(k)} \left(\vec{U}_n + \frac{1}{2}\vec{R}_2,\, t_n + \frac{1}{2}\tau\right), \\
R^{(k)}_4 &= \tau f^{(k)} \left(\vec{U}_n + \vec{R}_3,\, t_n + \tau\right),
\end{aligned} \tag{23}$$

where $\vec{U}_n$, $\vec{R}_1$, etc. are defined analogously to $\vec{u}$ in (22). As for a single ODE, an important consideration when solving systems of ODEs is the stability of the numerical method. Here the small deviation $\vec{\delta u}$ from the solution $\vec{u}$ of (22) satisfies an equation analogous to (9) where the coefficient $\partial_u f$ is replaced with the matrix whose (j,k)th entry is $\partial f^{(j)}/\partial u^{(k)}$. By diagonalizing this matrix (which is possible in the generic case), one finds that the component of $\vec{\delta u}$ "aligned along" the lth eigenvector of this matrix satisfies (10), with $\lambda \equiv \lambda_l$ being the corresponding eigenvalue. Thus, the stability analysis for systems of ODEs reduces to that for a single ODE, where λ may be complex even though the original equations are real-valued.

In this regard, systems which have purely imaginary eigenvalues when linearized near a stable equilibrium require special attention. An example of such a system that is typical in the study of population dynamics is the Lotka–Volterra predator-prey model:

$$\dot{u} = au - buv, \qquad \dot{v} = -cv + duv, \tag{24}$$

where u and v are the populations of the prey and predators and a, b, c, d are positive constants. Linearization of (24) near one of its equilibrium points, ($\bar{u} = c/d$, $\bar{v} = a/b$), yields a pair of imaginary eigenvalues $\pm i\sqrt{ac}$. Then the stability results presented above show that the explicit Euler method (2) is unstable for the Lotka–Volterra model; see Fig. 1. (Strictly speaking, the 2nd-order Heun method (15), (16) and even the 5th-order RK method used by Matlab's `ode45` are unstable, but their instability is much weaker than that of (2).) Moreover, the implicit Euler method (12), which is stable for this model, also produces a blatantly incorrect solution. Indeed, the point $\tau \cdot i\sqrt{ac}$ is strictly inside its stability region (see Fig. 1), and hence the numerical solution obtained by this method will spiral towards the equilibrium point. However, the exact analytical solution of (24) is known to orbit periodically about that point. As we show below, the following 1st-order-accurate modification of (2) yields a solution of (24) that stays near the exact periodic solution for all times:

$$U_{n+1} = U_n + \tau\, U_n(a - bV_n), \qquad V_{n+1} = V_n + \tau\, V_n(-c + dU_{n+1})\,. \tag{25}$$

The reason is that the stability region for this method can be shown to be the same as that of the leap-frog method (21), i.e. a segment along the imaginary $(\tau\lambda)$-axis (see Fig. 1). Since, as stated above, eigenvalues of the linearization of (24) at any point sufficiently near the equilibrium are almost purely imaginary, then the corresponding value $\tau\lambda$ is on the boundary of the stability region of method (25). Then small deviations from the exact solution (as well as from the equilibrium) will neither grow nor decay with time, but will oscillate around that solution. Thus, the important lesson to be drawn from this example is that the choice of the numerical method depends on the eigenvalues of the linearized system in question. That is, methods that work for systems with purely imaginary eigenvalues may not work for systems with real (and negative) eigenvalues, and vice versa. In passing, let us also mention that method (25) and the leap-frog method (24) are members of the family of symplectic methods which are designed specifically for stable numerical integration of models exhibiting oscillatory behavior without dissipation.

When dealing with an arbitrary system of ODEs, it is usually not possible to establish beforehand whether the eigenvalues of its linearization are purely imaginary or not. In such cases, one should use the RK methods (of order 4 or higher, if studying long-term evolution). Indeed, if any of the eigenvalues are not purely imaginary, one needs a method whose stability region extends into the left half of the complex $(\tau\lambda)$-plane. On the other hand, if there are purely imaginary eigenvalues, the corresponding values of $\tau\lambda$ must be on the boundary of the stability region, in order for the numerical solution to be close to the exact solution for all times (see above). From Fig. 2, one sees that the 4th- and 5th-order RK methods satisfy both of these requirements, since their right boundary is following a segment along the imaginary axis

very closely (although, as one can show, not exactly). To illustrate the applicability of high-order RK methods to models oscillatory behavior, we follow the evolution of a numerically computed quantity that is conserved by the exact solution of (24). In Fig. 3, we plot this quantity obtained with: the "simple" Euler method (2), the "symplectic" Euler method (25), and the RK method (19). For a fair comparison, we used $\tau = 10^{-3}$ for each of the first-order methods (2) and (25) and $\tau = 2 \cdot 10^{-1}$ for the 4th-order method (19). This choice of τ makes it clear that the observed poor performance of the "simple" Euler method is *not* due to its lower accuracy compared to the RK method, but due to the inferior stability of the former method. Indeed, the "symplectic" Euler method, whose accuracy is the same as that of "simple" Euler method but the stability region is "just right" for this problem, stays close to the exact solution (as can be shown, even for longer times than the RK method). < EDITORS: Fig. 3 near here. >

Stiff equations

Models where there are two (or more) disparate time scales give rise to so called stiff ODEs. A simple special case exhibiting such behavior is a system of two interacting species which, when considered independently of each other, evolve at their respective equilibria at very different rates. An example is a model (with exaggerated parameters) of interacting phytoplankton and zooplankton populations (P and Z), proposed by J.E. Truscott and J. Brindley in 1994:

$$\begin{aligned} \dot{P} &= \beta P(1-P) - Z\frac{P^2}{\nu^2 + P^2} \\ \dot{Z} &= \gamma Z\left(\frac{P^2}{\nu^2 + P^2} - \omega\right), \end{aligned} \tag{26}$$

where β, ν, γ, and ω are some positive constant parameters of the model. For $\beta = 2000$, $\nu = 0.05$, $\gamma = 0.4$, $\omega = 0.7$, and the initial condition $P(0) = Z(0) = 0.5$, the evolutions of P and Z are shown in Fig. 4. < EDITORS: Fig. 4 near here. > Note that when the two populations do not interact, P and Z tend to their equilibria (1 and 0) with rates proportional to the greatly different parameters β and $\gamma\omega$, respectively. Mathematically, these rates are proportional to the eigenvalues of the corresponging equations linearized about any "point" (P, Z) of the solution. In the presence of the interaction, the "memory" about these disparate time scales is reflected by the existence of both smooth and abrupt changes of P and Z. Let us emphasize that even though those abrupt changes (with the rate proportional to β) occur relatively rarely and "most of the time" the evolution is smooth, the two eigenvalues λ_1 and λ_2 of the linearized system (26) have magnitudes on the orders of $\beta \gg 1$ and $\gamma\omega = O(1)$ for all times. Then any explicit numerical method will require, as its stability condition, that $\tau \max\{|\lambda_1|, |\lambda_2|\} = \tau O(\beta)$ be less than a constant of order one. Thus, one needs to take steps $\tau = O(\beta^{-1}) \ll 1$ to resolve even the smooth motion occurring on the time scale $O((\gamma\omega)^{-1}) = O(1)$. A common approach to avoid such an inefficient use of computational resources is to use implicit methods (of a form more sofisticated than (12) and (14)) which, as stated earlier, have much greater stability regions than explicit methods and hence proceed in time with much larger steps.

Since the implementation of such methods requires the solution of (a system of) nonlinear algebraic equations, their programming takes much more effort than that of explicit methods. Fortunately, modern computational software have built-in commands for solving (systems of) stiff ODEs. For example, the solution shown in Fig. 4 was obtained by a Matlab script shown in Fig. 5. (The nonstiff solver `ode45` for this problem would run about 50 times slower and for larger values of β, it would simply fail.) Even though the example above uses unrealistic values of parameters, it still illustrates the issue which occurs in many known systems of a large number of coupled equations (for example, equations describing chemical reactions among trace gases in the atmosphere). < EDITORS: Fig. 5 near here. >

The periodic solutions in the plankton model (26) and Lotka–Volterra equations are different in the following regard. Any (meaningful) initial condition in the plankton model will be attracted to the unique periodic solution shown in Fig. 4, while for the Lotka–Volterra model, any initial condition will be repeated periodically, i.e. a continuum of periodic solutions exists. Correspondingly, the respective models are often referred to as excitable and conservative. Conservative models with four and greater even number of interacting species may have imaginary eigenvalues of disparate magnitude. (According to our definition, such systems would be also called stiff, but the convention is to use that name only for systems with $\mathrm{Re}\lambda < 0$.) Therefore, a suitable numerical method must have a section of its stability boundary following the imaginary axis. The corresponding Matlab's stiff solver is `ode23t`. Its performance can be tested on the following simple model:

$$\dot{u} = 1000i\, u + e^{it}\,, \tag{27}$$

which has no ecological interpretation but serves to illustrate the above point. Indeed, the "free" solution of (27) oscillates 1000 times faster than the "forced" one, hence two disparate rates of conservative evolution are present. Other stiff and nonstiff Matlab's solvers do not perform satisfactorily for this problem. Thus, the conclusion drawn from this example is similar to that stated earlier, namely: Methods that can be applied to nonconservative systems (such as (26)) may not work for conservative ones, and vice versa.

Further Reading

Chandra, P.K. and Singh, R.P. (1995). *Applied numerical methods for food and agricultural engineers*. Boca raton, FL: CRC Press.

Gerald, C.F. and Wheatley, P.O. (1989). *Applied numerical analysis* (4th edn.). Reading, MA: Addison Wesley.

Lee, H.J. and Schiesser, W.E. (2003). *Ordinary and partial differential equation routines in C, C++, Fortran, Java, Maple, and MATLAB*. Boca Raton, FL: Chapman & Hall / CRC.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. (1992). *Numerical recipes in Fortran 77* (2nd edn.). Cambridge, UK: Cambridge University Press; Press, W.H.,

Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., Metcalf, M. (1996). *Numerical recipes in Fortran 90* (2nd edn.). Cambridge, UK: Cambridge University Press; Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. (1992). *Numerical recipes in C* (2nd edn.). Cambridge, UK: Cambridge University Press.

Sauer, T. (2006). *Numerical analysis*. Boston, MA: Pearson/Addison Wesley.

Shampine, L.F. (2005). Error estimation and control for ODEs, *SIAM Journal of Scientific Computing* **25**, 3–16.

Stanoyevitch, A. (2004). *Introduction to numerical ordinary and partial differential equations using MATLAB*. Hoboken, NJ: Wiley-Interscience.

See also in this volume:

Numerical methods for distributed models

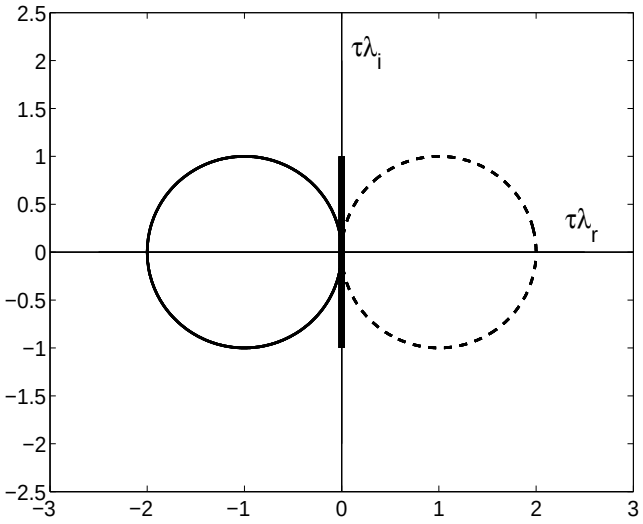

Figure 1: Stability region boundaries of methods (2) (thin solid), (12) (dashed), and (21) (thick solid). Note that for (12), the stability region is *outside* the dashed circle.

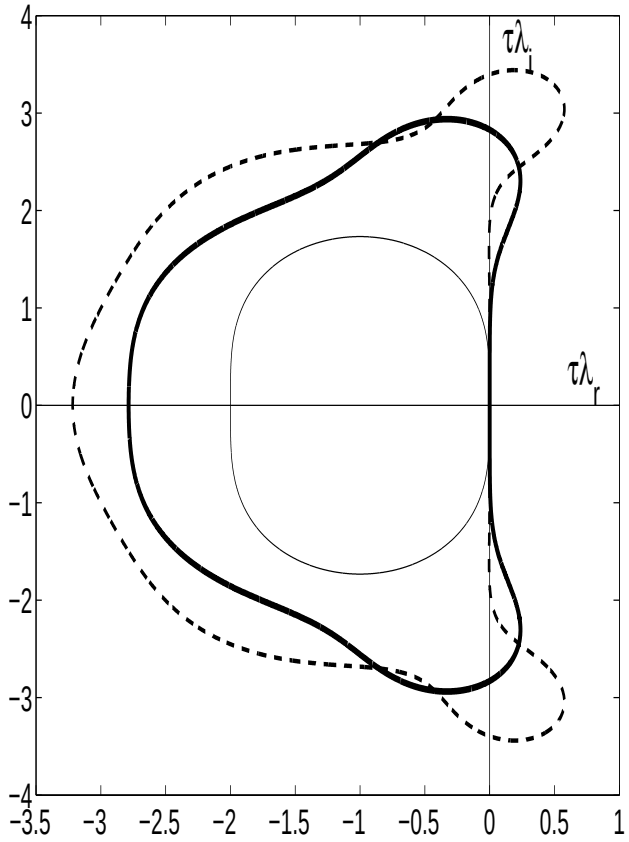

Figure 2: Stability region boundaries of methods (15), (16) (thin solid), (19) (thick solid), and the 5th-order RK method (dashed).

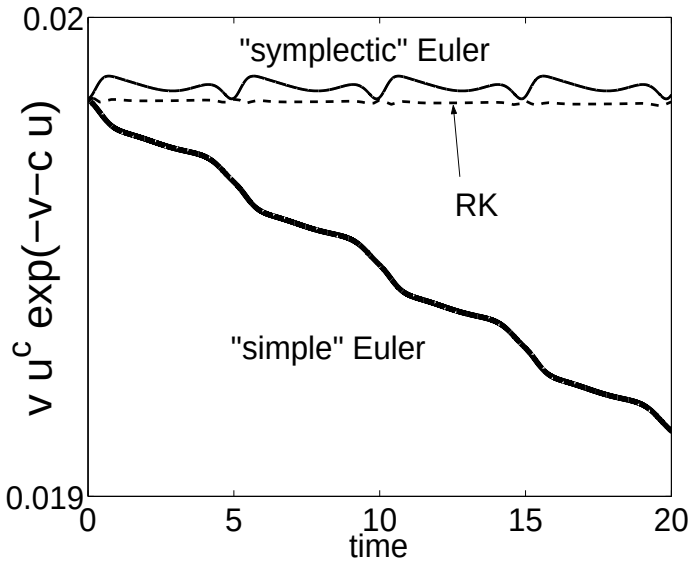

Figure 3: Evolution of quantity $Q = vu^c \exp[-v - cu]$, which is conserved for the exact solution of the Lotka–Volterra model with $a = b = 1$ and $d = c$. The initial condition is $u_0 = v_0 = 2$, and $c = 2$. The results are shown for methods (2) ("simple" Euler), (25) ("symplectic" Euler), and (12) (4th-order RK). Values of the step size are indicated in the text. The smaller the deviation of $Q(t)$ from $Q(0)$, the better the method performs.

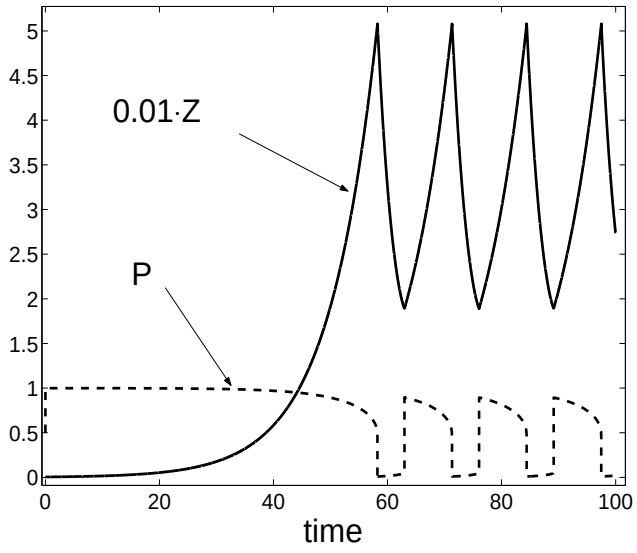

Figure 4: The solution of the stiff model (26) with the parameters indicated in the text.

```
global   beta nu gamma omega
                % This allows one to communicate the
                %  parameter values to code plankton.m

tspan=[0 100];  % computational time window
y0=[0.5; 0.5];      % initial condition for u and du/dt
beta=2000;  nu=0.05;  gamma=0.4;  omega=0.7;

options=odeset('RelTol',10^(-3),'AbsTol',10^(-6));
                % These are default values;
                %  the command is shown for
                %  illustration purposes only.
[t,y]=ode15s(@plankton,tspan,y0,options);
plot(t,y(:,1))

% ------------------------
% This is a separate function code
%  defining the equations of the model.

function dydt=plankton(t,y)

global beta nu gamma omega

dydt=zeros(size(y));
                % preallocate column vector dydt
                %  to speed up the computation

p2=y(1,:).^2;  auxiliary=p2./(nu^2+p2);
dydt(1,:)=beta*y(1,:).*(1-y(1,:))-y(2,:).*auxiliary;
dydt(2,:)=gamma*y(2,:).*(auxiliary-omega);
```

Figure 5: The Matlab codes used to obtain the solution shown in Fig. 4.