Numerical methods for distributed 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 partial differential equations of the reaction-advection-diffusion type are presented. The use of implicit (e.g., Crank-Nicolson) and semi-implicit methods is emphasized as being more time-efficient than that of explicit methods. The main kinds of boundary conditions are considered. Briefly discussed are operator-splitting and spectral methods, as well as methods for equations with two spatial dimensions and for hyperbolic problems. Relevant built-in commands of Matlab are presented and their counterparts in other programming languages are briefly mentioned.

Keywords: Numerical solution of differential equations, spatially-extended models, age-structured models, reaction-advection, reaction-diffusion, and advection-diffusion 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.

Partial differential equations (PDEs) in ecological modeling

Such equations arise whenever the quantities of interest vary not only with time but also with another variable. For example, in age-structured models, this variable is the age of a given population subroup. When there are spatial interactions in the model caused by diffusion and convection, the other variable is the spatial coordinate (or coordinates). In what follows we focus on those models where the effect of diffusion is on the order of, or greater than, the effect of convection. These models are described by so called parabolic equations. Models where convection dominates diffision (this case includes the age-structured models) pertain to a mathematically different type of problems called hyperbolic, or wave, equations; they will only be mentioned briefly at the end of this article. To further narrow the scope of this article, we will consider only finite-difference and elementary spectral numerical methods. The reader can find accessible expositions of finite-element methods in, e.g., the books by Stanoyevich and Chandra and Singh. The advantage of those methods over the finite-difference ones is that they allow one to use a non-uniform spatial discretization and also to account for complex shapes of two- and three-dimensional spatial domains. The price one pays for this advantage is a more involved programming.

Explicit methods for reaction-advection-diffusion equations

The equation whose solution we will focus on is

$$\dot{u} = D(u,x,y)\partial_x^2 u + g(u,x,t)\partial_x u + f(u,x,t), \qquad u(x,t_0) = u_0(x) \tag{1}$$

where: t and x are the time and space variables and $\dot{u} \equiv \partial_t u$. (A generalization of (1) to two spatial dimensions will be considered later.) The three terms on the r.h.s. of (1) are associated with diffusion, advection, and reaction, respectively. To illustrate the main issues of numerical solution of (1), we first consider its simpler version — the Heat equation — where we retain only the diffusion term and take the diffusion coefficient D to be a constant:

$$\dot{u} = D\partial_x^2 u, \qquad u(x,t_0) = u_0(x)\,; \tag{2}$$

$$u(0) = 0, \quad u(X) = 0\,. \tag{3}$$

The last two equations are called boundary conditions; such (or similar) conditions must always be supplied for a PDE. Boundary conditions that fix the value of the unknown, as in (3), are called Dirichlet boundary conditions. Boundary conditions of other types will be considered later. Note that setting the boundary conditions (3) to zero does not restrict the generality of our treatment. Indeed, for the more general conditions, $u(0) = a(t)$, $u(X) = b(t)$, one defines a new variable $w = u - [(X-x)a(t) + xb(t)]/X$, which satisfies (2) and (3).

We discretize $t_n = t_0 + n\tau$ and $x_m = 0 + mh$, where τ and h are the time step and the mesh size, and $n = 0, 1, 2, \ldots$ and $m = 0, 1, \ldots, X/h \equiv M$. Denote $u_m^n \equiv u(x_m, t_n)$ and U_m^n to be,

respectively, the exact and the numerical solutions at node (x_m, t_n). Replacing

$$\begin{aligned} \dot{u} \quad &\rightarrow \quad (U_m^{n+1} - U_m^n)/\tau + O(\tau) \\ \partial_x^2 u \quad &\rightarrow \quad (U_{m+1}^n - 2U_m^n + U_{m-1}^n)/h^2 + O(h^2)\,, \end{aligned} \tag{4}$$

one obtains the finite-difference approximation to (2):

$$U_m^{n+1} = rU_{m+1}^n + (1-2r)U_m^n + rU_{m-1}^n\,, \qquad U_0^n = 0, \quad U_M^n = 0\,, \tag{5}$$

where

$$r = \frac{D\tau}{h^2}\,. \tag{6}$$

The meaning of the notation $O(\tau^p)$ was explained in the companion article on ODEs. Equation (5) is the counterpart of the explicit Euler method for ODEs. As follows from (4), its accuracy is $O(\tau) + O(h^2)$. (However, for an initial condition $u_0(x)$ that is either discontinuous or has a discontinuous slope, the spatial accuracy is reduced: the smaller r in (6), the closer the accuracy to the maximum possible value $O(h^2)$.) The numerical solution at node (x_m, t_{n+1}) can thus be found from (5 and the initial condition $U_m^0 = u_0(x_m)$ if one knows the solution at nodes (x_m, t_n) and $(x_{m\pm1}, t_n)$. These four nodes form a so called stencil for scheme (5), as shown schematically in Fig. 1. < EDITORS: Fig. 1 near here. >

Stability analysis

As in the case of ODEs (see the companion article), the numerical method must be not only accurate but also stable in order to converge to the exact solution when τ and h tend to zero. The following von Neumann analysis is the standard tool to test stability of a numerical method. First, we note that if U_m^n and V_m^n are two slightly different (due to, e.g., different initial conditions) solutions of (5), then the error $\epsilon_m^n = V_m^n - U_m^n$ satisfies the same Eq. (5) (since the latter is linear). This error can be expanded into a series of spatial Fourier harmonics:

$$\epsilon_m^n = \sum_l c_l(n)\, \exp[ik_l x_m], \qquad i \equiv \sqrt{-1}\,, \tag{7}$$

where $c_l(n)$ are amplitudes of the harmonics and k_l are inversely proportional to the spatial oscillation periods of the harmonics. (Note that while the variable ϵ_m^n is real, both the exponentials and their amplitudes c_l are complex numbers; this is a standard mathematical procedure that does not restrict the validity of the approach.) The "slowest" harmonics have $k_l \approx 0$ are are changing very slowly with x. The "fastest" harmonic has $k_l \approx \pi/h$, which corresponds to the shortest possible oscillation period of $2h$; see Fig. 2. < EDITORS: Fig. 2 near here. > Next, one assumes that the amplitudes $c_l(n)$ depend exponentially on the number of iterations: $c_l(n) = \rho^n$. Then if the subsequent analysis finds that $|\rho| \leq 1$, then the method is stable, and it is unstable otherwise. The substitution of (7) into (5) yields the stability condition to be $|\rho| = |1 - 2r + 2r\,\cos(k_l h)| \leq 1$, which with the above bounds for k_l ($\min k_l = 0$ and $\max k_l = \pi/h$) gives the stability condition for (7):

$$r \;\equiv\; D\tau/h^2 \;\leq\; 1/2\,. \tag{8}$$

If this condition is violated (due to taking either τ "too large" or h "too small"), the fastest spatial harmonic begins to grow exponentially, and the numerical solution soon becomes "drowned" in a high-frequency noise.

From (8) one has $\tau = O(h^2)$. Then the accuracy of method (5) can be expressed only in terms of the mesh size h: $O(\tau) + O(h^2) = O(h^2) + O(h^2) = O(h^2)$. This leads to the following observation. As we noted above, method (5) is the counterpart of the explicit Euler method for ODEs (see Eq. (2) of the companion article). Other explicit methods may have a higher-order accuracy in the time step; e.g., the PDE counterpart for (2) of the Heun method (Eqs. (15), (16) of the previous article) would have accuracy $O(\tau^2)$. However, this would not increase the overall accuracy of such a method. The reason is that the stability condition for this method is also given by (8). Then, similarly to the above, the overall accuracy of the counterpart of the Heun method for (5) is $O(\tau^2) + O(h^2) = O(h^4) + O(h^2) = O(h^2)$.

Implicit methods for the Heat equation

Stability condition (8) forces one to use a rather small time step, which for integration over large times may take a considerable computational time. (With modern computers, this is rarely an issue when the PDE involves only one spatial coordinate, but for problems with two (or three) spatial coordinates, this issue is still significant.) Therefore, methods whose stability conditions would be less restrictive than (8) (or even absent) are of considerable interest. A famous example is the Crank–Nicolson (CN) method, obtained from the following discretization of (5):

$$\frac{U_m^{n+1} - U_m^n}{\tau} = \frac{D}{2}\left[\frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{h^2} + \frac{U_{m+1}^{n+1} - 2U_m^{n+1} + U_{m-1}^{n+1}}{h^2}\right]. \tag{9}$$

This method is stable unconditionally (i.e., for any τ), and its accuracy is $O(\tau^2) + O(h^2)$, as we will explain below. Its stencil is shown in Fog. 3 by the six grey circles. < EDITORS: Fig. 3 near here. > ¿From (9) one can see that U_m^{n+1} cannot be determined in isolation. Rather, one has to determine the vector of the unknowns on the entire $(n+1)$th time level. Introducing notations

$$\vec{\mathbf{U}}^n = \begin{pmatrix} U_1^n \\ U_2^n \\ \cdot \\ U_{M-2}^n \\ U_{M-1}^n \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} -2 & 1 & 0 & \cdot & \cdot & 0 \\ 1 & -2 & 1 & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & 1 & -2 & 1 \\ 0 & \cdot & \cdot & 0 & 1 & -2 \end{pmatrix} \tag{10}$$

one can rewrite the CN method (9) as a matrix equation:

$$\left(I - \frac{r}{2}A\right)\vec{\mathbf{U}}^{n+1} = \left(I + \frac{r}{2}A\right)\vec{\mathbf{U}}^n, \tag{11}$$

where I is the $M \times M$ unit matrix and r is defined in (8).

The number of arithmetic operations required to solve a generic $M \times M$ matrix equation (i.e., a system of M linear equations with M unknowns) is on the order of M^3. Thus, if the spatial domain is discretized by, say, 100 mesh points, solving such a generic $M \times M$ matrix equation would take about one million operations per time step. This is many orders

of magnitude greater then the operations count for method (5), which requires only about $4M$ operations per time step. However, the matrix (on the l.h.s.) of the CN method (11) is not generic. Rather, it is tridiagonal, meaning that only its entries on the main diagonal and the two adjacent sub- and super-diagonals are nonzero (see (10)). For such matrices, it is possible to solve (11) using only about $8M$ operations, a tremendous improvement compared to the generic count of about M^3. The corresponding solution method is called the Thomas algorithm (invented in the 1940's) and is discribed in most textbooks on numerical solution of PDEs or on numerical analysis. This method is also rather easy to program. Thus, the CN method offers a considerable computational time saving compared to the explicit method (5), because the time step in the CN method is no longer restricted by (8) to ensure stability.

However, the time step in the CN method still needs to be about, or smaller than, the mesh size h to ensure the accuracy $O(h^2)$. This is especially important if the initial condition u_0 has sharp corners or is discontinuous. In such a case, it has been found empirically that nonsmooth features of $u_0(x)$ "diffuse away" (as they do in the exact solution) and do not contaminate the numerical solution when $\tau < h/(3\sqrt{D})$. Also, the smaller r, the faster these nonsmooth features "diffuse away" in the numerical solution. Alternatively, for a nonsmooth u_0, one can use another method:

$$\frac{3}{2}\,\frac{U_m^{n+1}-U_m^n}{\tau} - \frac{1}{2}\,\frac{U_m^n-U_m^{n-1}}{\tau} \;=\; D\,\frac{U_{m+1}^{n+1}-2U_m^{n+1}+U_{m-1}^{n+1}}{h^2}\,, \tag{12}$$

which has the same accuracy and the stability property as the CN method, but smoothes out nonsmooth initial conditions much more successfully (i.e., for larger r.) (The derivation of this formula uses the same idea as the derivation of the two-step method (20) in the companion article on ODEs.) Method (12) is to be started by computing U_m^1 by, say, the CN method. To program the rest of (12), one moves all of the U^{n+1}'s on the l.h.s. and all the other terms on the r.h.s. and then rewrites this as a matrix equation analogous to (11). The matrix on the l.h.s. is tridiagonal and hence can be inverted time-efficiently using the Thomas algorithm.

We now briefly explain why the accuracy of the CN method is $O(\tau^2)+O(h^2)$; this will show how (11) can be generalized for the more general equation (1). First, note that for any function $u(t)$, different discretization schemes approximate the derivative $\dot{u}$ with different accuracy. For example,

$$\dot{u}(t) = \frac{u(t+\tau)-u(t)}{\tau} + O(\tau), \tag{13}$$

$$\dot{u}(t) = \frac{u(t+\tau)-u(t-\tau)}{2\tau} + O(\tau^2), \tag{14}$$

as can be shown using the Taylor expansion of u near time t. That is, the approximation at point t is more accurate if one uses the data points located symmetrically on both sides of t. Note then that points t and $t+\tau$ are located symmetrically about the point $t+\frac{1}{2}\tau$. Therefore, a formula similar to (14) approximates $\dot{u}$ at that point with accuracy $O(\tau^2)$:

$$\dot{u}\left(t+\frac{\tau}{2}\right) \;=\; \frac{u(t+\tau)-u(t)}{2\,(\tau/2)} + O(\tau^2). \tag{15}$$

Therefore, the quotient appearing on the l.h.s. of (11) approximates $\dot{u}$ with accuracy $O(\tau^2)$ about the virtual node at $(x_m, t_n+(\tau/2)\,)$, shown in Fig. 3 with a cross. Finally, similarly to

(14) and (15) it can be shown that,

$$\partial_x^2 u\left(t+\frac{\tau}{2}\right) \;=\; \frac{\partial_x^2 u(t+\tau)+\partial_x^2 u(t)}{2}+O(\tau^2). \tag{16}$$

Combining (16) with the second equation in (4), one has that the r.h.s of (11) approximates $\partial_x^2 u$ at the same virtual node with accuracy $O(h^2)+O(\tau^2)$. Thus, the overall accuracy of (11) is $O(\tau^2)$ (from the l.h.s.) plus $O(h^2)+O(\tau^2)$ (from the r.h.s.), resulting in the overall value $O(\tau^2)+O(h^2)$.

Implicit and semi-implicit methods for (1) and for similar coupled systems

Let us first show how the idea of the previous paragraph can be used to obtain a numerical method for a single equation (1) where the coefficients D, g, and f depend only on x and t (i.e., do not depend on the unknown u). Equations with this property are referred to as linear. Generalizations to nonlinear equations and systems of linear or nonlinear coupled equations will be considered later. The second-order accurate approximations to the terms in (1) at the virtual node $(x_m, t_n+(\tau/2)\,)$ are produced by the following discretization schemes:

$$\begin{aligned}
\dot{u} \qquad &\to \quad \frac{U_m^{n+1}-U_m^n}{\tau}\,,\\
D(x,t)u_{xx} \quad &\to \quad \frac{1}{2}\left(D_m^n\frac{U_{m+1}^n-2U_m^n+U_{m-1}^n}{h^2}+D_m^{n+1}\frac{U_{m+1}^{n+1}-2U_m^{n+1}+U_{m-1}^{n+1}}{h^2}\right),\\
g(x,t)u_x \quad &\to \quad \frac{1}{4h}\left(g_m^n(U_{m+1}^n-U_{m-1}^n)+g_m^{n+1}(U_{m+1}^{n+1}-U_{m-1}^{n+1})\right),\\
f(x,t)u \quad &\to \quad \frac{1}{2}\left(f_m^n U_m^n+f_m^{n+1}U_m^{n+1}\right).
\end{aligned} \tag{17}$$

Putting these terms together produces an unconditionally stable (i.e., with no restriction on τ for the stability) method with the overall accuracy $O(\tau^2)+O(h^2)$. Naturally, it is assumed that the coefficients D, g, and f change little over distances of order h and times of order τ. Also, the above discretizations are not unique in the sense that replacing both D_m^n and D_m^{n+1} with $D_m^{n+\frac{1}{2}} = D(x_m, t_n+(\tau/2))$ (and similarly for g and f) yields another scheme with the same accuracy and the stability property as (17).

When generalizing the previous technique to the case where D, g, and f depend on the u, one faces the problem that the r.h.s.'s in (17) become nonlinear functions of the unknown variables $\vec{\mathbf{U}}^{n+1}$ (see (10)) on the $(n+1)$th time level. (If one uses $D_m^{n+\frac{1}{2}}$ instead of D_m^{n+1} as suggested above, one still needs to find $U_m^{n+\frac{1}{2}n}$.) Solving systems of nonlinear algebraic equations requires considerably more effort than solving their linear counterparts. A common approach is to do so by the Newton-Raphson method, described in textbooks on numerical analysis. This approach needs to be used when one wants to obtain an unconditionally stable method with the overall accuracy $O(\tau^2)+O(h^2)$ and, in addition, either of the following holds: (i) D depends on the unknown u; or (ii) The reaction term f is stiff in the sense described in the companion article

on ODEs. (The latter case may take place not for the single equation (1) but for two or more coupled equations of that form.)

However, when both the diffusion coefficient D does not depend on the unknown variable (although it may still depend on x and t) and the reaction term is not stiff, a simpler approach can be used. Its idea is to use the CN method (or any method with analogous accuracy and stability; e.g., (12)) to discretize the diffusion term, while using an explicit method to discretize the other two terms on the r.h.s. of (1). Indeed, since the diffusion term is computed by the unconditionally stable CN method, then the restriction on the time step is imposed by the explicit calculation of the other two terms, which requires only that

$$\tau < \min[h/|g|_{\max},\, 2/|f|_{\max}] \tag{18}$$

and is much less restrictive than (8) when h is small. (In fact, as we noted earlier, the condition $\tau < \text{const} \cdot h$ needs to hold just to ensure good accuracy of the method.) Let us rewrite (1) as

$$\dot{u} = \mathcal{I} + \mathcal{E}, \tag{19}$$

where the evolutions associated with $\mathcal{I}$ and $\mathcal{E}$ need to be computed implicitly and explicitly, respectively. (In the case of (1), $\mathcal{I} = D\partial_x^2 u$ and $\mathcal{E} = g\partial_x u + f$.) Then a method that has the accuracy $O(\tau^2) + O(h^2)$ and the stability restriction of the form (18) is

$$\frac{U_m^{n+1} - U_m^n}{\tau} = \left(\gamma \mathcal{I}_m^{n+1} + \left[\frac{3}{2} - 2\gamma\right] \mathcal{I}_m^n + \left[\gamma - \frac{1}{2}\right] \mathcal{I}_m^{n-1}\right) + \left(\frac{3}{2}\mathcal{E}_m^n - \frac{1}{2}\mathcal{E}_m^{n-1}\right), \tag{20}$$

where $1/2 \le \gamma \le 1$ and, in our example,

$$\mathcal{I}_m^n = D_m^n \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{h^2}, \qquad \mathcal{E}_m^n = g_m^n \frac{U_{m+1}^n - U_{m-1}^n}{2h} + f_m^n, \tag{21}$$

etc. Method (20) is an example of a semi-implicit, or IMEX, method. It is a two-step method and, as (12 above, it needs to be started by computing U_m^1 by a single-step method (e.g., by (20) with $\gamma = 1/2$ and the second parentheses on the r.h.s. being replaced with $\mathcal{E}_m^n$). When $\gamma = 9/16$, method (20) is most (among all γ's) efficient for smoothing our nonsmooth initial confitions, while when $\gamma = 3/4$, its stability boundary is most extended along the imaginary $(\tau\lambda)$-axis. As shown in the companion article on ODEs, this feature of the method is essential for stability when the explicitly treated terms in (1) describe no or little dissipation. Mathematically, this is the case when, e.g., the advection coefficient g is on the order of or greater than the reaction term f.

In connection with the last remark, let us also note that in that case, one should not use IMEX methods where the $\mathcal{E}$-term is computed by any method in the Runge–Kutta family. This may sound surprising given that the latter methods were advocated in the companion article on ODEs precisely because they have part of their stability region boundary extended along the imaginary $(\tau\lambda)$-axis, which is required for successful numerical solution of ODEs with no or little dissipation. However, when a Runge–Kutta method is used to compute the $\mathcal{E}$-term in an IMEX method, the stability region of such an IMEX method turns out to be close to that of the explicit Euler method, and the latter region's boundary does not extend along the imaginary $(\tau\lambda)$-axis.

In many cases one needs to deal not with a single reaction-advection-diffusion equation but with a system of coupled such equations. However, in most applications, the diffusion terms are decoupled. That is, a term proportional to $\partial_x^2 u_j$ enters only into the equation for $\dot{u}_j$ but not into the equations for other unknown variables u_i with $i \neq j$. In such a case, one can straightforwardly generalize method (20) (or any other IMEX method). Indeed, moving the $\mathcal{I}_m^{n+1}$-term to the l.h.s. of the equation for the jth variable yields a tridiagonal matrix multiplying only that variable and not involving other unknown variables at the $(n+1)$th time level. The r.h.s. of the equation does involve all of the variables (because they are coupled via the $\mathcal{E}$-term), but they are evaluated at the earlier time levels and therefore have been already calculated at previous steps. Thus, each of the equations can be solved time-efficiently by the Thomas algorithm. The above approach can be used for both linear and nonlinear equations of form (1). For linear equations only, the block Thomas algorithm (described in more advanced textbooks or in the Internet resources) can be used as an alternative. Finally, let us note that if terms $D_{ij}\partial_x^2 u_j$ enter into the equation for $\dot{u}_i$ with $i \neq j$, where all of the D_{ij} do not depend on x and u (but are allowed to depend on t), the $\mathcal{I}$-part of such a system can be made uncoupled by a change of the unknown variables that diagonalizes matrix D_{ij}. The corresponding methods are considered in all textbooks on Linear Algebra.

Let us note that Matlab has a built-in solver for coupled equations of form (1) in one spatial dimension and with arbitrary D, g, and f. Matlab's setup allow one to also handle radially symmetric solutions in regions with circular symmetry. The user must provide the set of discretization points in the spatial interval and also code in the coefficients D, g, and f and the boundary and initial conditions. See Matlab's help for `pdepe`.

Operator-splitting methods

Suppose one has an evolution equation of a general form:

$$\dot{u} = A_1 + A_2, \tag{22}$$

where each of A_1 and A_2 may depend on x, t, u, $\partial_x u$, etc. The criterion by which the r.h.s. of (22) has been broken down into the two terms is this: Each of the auxiliary problems

$$\dot{u} = A_k, \qquad k = 1, 2 \tag{23}$$

can be solved easily. Then, to approximate the solution of (22), one solves the two individual equations (23) in sequence over each time step. The global error generated by this splitting is $O(\tau)$, in addition to any errors that may be introduced when solving each of the (23). More accurate versions of this method are also available. However, we do not advocate for operator-splitting methods because of their stability properties. Namely, although for linear equations such methods are unconditionally stable, for nonlinear equations, in general, the stability of operator-splitting methods is determined by a condition similar to (8), even if each of Eqs. (23) is solved exactly. Hence, in such cases, operator-splitting methods offer no advantage over explicit methods. Operator-splitting methods find their primary use in solving (nonlinear) wave equations without dissipation, which, however, do not normally occur in ecological models.

Flux and mixed boundary conditions

We now show a common trick that can be used to construct methods of the same accuracy as above when boundary conditions other than (3) are used. Namely, while (3) specify the value of the unknown at the boundaries (recall that it was set to 0 without loss of generality), in some problems it may be the flux of u, or a linear combination of u and its flux, that are to be specified. Since the flux of u is proportional to $\partial_x u$, it is the calculation of $\partial_x u$ at the boundary with accuracy $O(h^2)$ that requires special treatment. Thus, consider the condition at, say, the left boundary, which replaces the first condition in (3):

$$\partial_x u(0,t) + q(t)u(0,t) = b(t)\,, \tag{24}$$

where $q(t)$ and $b(t)$ are known for any t. When $q(t) \equiv 0$, this is called a Neumann boundary condition, and when $q(t) \neq 0$, a mixed, or Robin, boundary condition. One discretizes (24) by introducing a fictitious node at $x = -h$:

$$\frac{U_1^n - U_{-1}^n}{2h} + q(t_n)U_0^n = b(t_n)\,, \tag{25}$$

where $U_{-1}^n = u(-h, t_n)$; the accuracy of this approximation is $O(h^2)$ (compare with (14)). This one equation introduces two new unknowns, U_0^n and U_{-1}^n. Therefore, one more equation involving these quantities must be supplied. Such an equation is given by the first equation in (5) with $m = 0$ for explicit methods and by (9) with $m = 0$ for implicit ones. For explicit methods, this concludes the treatment of (24). For implicit methods, one more step is needed. Recall that to render the solution of the matrix equation (11) time-efficient, the matrix on its l.h.s. must be tridiagonal. If (25) and (9) with $m = 0$ are both included separately into the counterpart of (11) arising for boundary conditions (24), the resulting matrix is not tridiagonal. To circumvent this problem, one should, instead of including the aforementioned two equations separately, solve (25) for U_{-1}^n and substitute the result into (9) with $m = 0$. Then the matrix in question becomes tridiagonal and the analogue of (11) can be solved time-efficiently.

Elementary spectral methods

The use of such methods for solving evolutionary PDEs has become increasingly common due to the availability of the required built-in commands in major computational software such as Matlab and Fortran. The idea of the method is based on the following property of the Fourier transform of a function. If $\hat{f}(k)$ is the Fourier transform of $f(x)$ with respect to x, then $(-i\,k)^m \hat{f}(k)$ is the Fourier transform of $\partial_x^m f(x)$. Here $i = \sqrt{-1}$ and k is the parameter of the Fourier transform, sometimes referred to as the wavenumber or frequency. For example, in Matlab, `fft` and `ifft` are the commands of the Fourier transform and the inverse Fourier transform. < 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. > (The entire acronym stands for Fast Fourier transform; similar commands are available in Fortran and C.) Then if `k` is the transform parameter properly defined within the code, then $\partial_x^2 u$ can be computed for all x at once by a short command: `real( ifft(-(k.^2).*fft(u)) )`. (Here the command `real` discards a very small imaginary part of the answer which arises from the round-off error, and the period before the exponentiation and multiplication symbols is part of Matlab's syntax for array multiplication.) This command can be also used in, say, IMEX methods. For example, method (20) can be programmed as follows:

```
K2=k.^2;
u=real( ifft( ( (1-T*(3/2-2*G)*K2).*fft(u) - T*(G-1/2)*K2.*fft(uold) ...
  + T*fft(3/2*E-1/2*Eold) )./(1+T*G*K2) ) );
```

where `T`$\equiv \tau$, `G`$\equiv \gamma$, and `uold` and `Eold` are the stored values of U and $\mathcal{E}$ at t_{n-1}. If $\partial_x u$ is present in $\mathcal{E}$, it can also be computed using `fft` and `ifft`, as explained above. In addition to the coding simplicity, advantages of the above approach are that it is (i) fast (requires only on the order of $M \log_2 M$ operations for each of the `fft` and `ifft`, where M is the number of the discretization points in $\vec{\mathbf{U}}$) and (ii) has very high accuracy in h provided that the solution is smooth (see below).

The main limitation of this approach is that, by the very nature of discrete Fourier transform, the discretized functions on which it is implemented must be periodic (or very close to periodic). This means that not only the values of the function but also its slopes at the end points of the interval must match each other. Thus, boundary conditions (3) are not periodic despite the fact that $u(0,t) = u(X,t)$; the reason is that $\partial_x u(0,t)$ and $\partial_x u(X,t)$ are not enforced to be equal. (For example, $\sin(\pi x/X)$ satisfies (3), yet is not periodic on $[0,X]$ since its slopes at $x=0$ and $x=X$ are opposite.) If discrete Fourier transform is used to compute a derivative of a function that is not periodic in the above sense, small-period but finite-amplitude oscillations arise at the end points which then propagate inside the interval and destroy the numerical solution. (In general, if $\partial_x^{m-1} u$ is continuous but $\partial_x^m u$ is discontinuous either at the end points or inside the interval, then the spatial accuracy of the solution is $O(h^m)$.)

Problems with periodic boundary conditions rarely arise in ecological modeling. Yet, the use of Fourier transform and its variant (see below) can be justified in the following two cases. First, the solution may be localized within a central part of the interval and be virtually zero (or tend to the same constant value) in some extended regions near the end points. Such a solution is very close to periodic. Second, the problem may have zero flux boundary conditions, i.e. (24) with $q(t) = b(t) \equiv 0$; also, it may asymptotically tend to different constants at the end points (as, for example, the travelling wave in the Fisher–Kolmogorov–Petrovsky–Piskunov equation). The (real-valued) variant of discrete Fourier transform called discrete cosine transform can be used on such functions. The corresponding commands `dct` and `idct` are included in Matlab's version 7.0.0 and beyond; earlier versions have them in the Signal processing toolbox. The key difference from the Fourier transform commands is that only even derivatives can be computed with the cosine transforms; e.g., $\partial_x^{2m} u$ is computed as `idct( (-k.^2).^m .*dct(u) )`, provided that u satisfies the zero flux boundary conditions.

Problems with 2 spatial dimensions

Such problems are considerably more difficult than those with one dimension. First, if the spatial region does not have rectangular or circular symmetry, the numerical modeling of the problem requires the use of finite element methods. In circular regions, the problem needs to be reformulated in polar coordinates. Below we consider only methods for rectangular regions. As earlier, we explain the main issues using the two-dimensional Heat equation

$$\dot{u} = \partial_x^2 u + \partial_y^2 u \tag{26}$$

with conditions along the boundary of the rectangle $0 \le x \le X$, $0 \le y \le Y$ which will be specified in each particular case. Generalizations to more realistic equations, such as a two-dimensional counterpart of (1), can be made along the lines considered above. Withough loss of generality we assume that the mesh size along the x- and y-directions is the same and equals h; the numerical solution at node $(x_m = mh,\ y_l = lh,\ t_n = n\tau)$ is denoted $U^n_{m,l}$, $0 \le m \le M$, $0 \le l \le L$. Two time levels for such a discretization are shown in Fig. 4, with the boundary being shown by open circles. < EDITORS: Fig. 4 near here. >

First, the explicit Euler method for (26) is:

$$U^{n+1}_{m,l} = (1-4r)U^n_m + r(U^n_{m-1,l} + U^n_{m+1,l} + U^n_{m,l-1} + U^n_{m,l+1})\,, \tag{27}$$

where r is defined in (6) with $D = 1$. The stencil of this method is shown in Fig. 5. < EDITORS: Fig. 5 near here. > The accuracy of this method is $O(\tau) + O(h^2)$ and its stability condition is

$$r \le 1/4\,. \tag{28}$$

Any of the Dirichlet, Neumann, or mixed boundary conditions can be used with this method similarly to how it was done in one dimension. In addition, if either periodic or zero flux (a particular case of Neumann) boundary conditions can be used, one can use two-dimensional built-in commands to approximate the r.h.s. of (26); in Matlab such commands are `fft2` and `dct2`. The stability condition in this case is more restrictive:

$$r \le 1/\pi^2\,. \tag{29}$$

A naive generalization of the CN method has the form:

$$\begin{aligned} & (1+2r)U^{n+1}_{m,l} - \frac{r}{2}\left(U^{n+1}_{m+1,l} + U^{n+1}_{m-1,l}\right) - \frac{r}{2}\left(U^{n+1}_{m,l+1} + U^{n+1}_{m,l-1}\right) \\ = \;& (1-2r)U^n_{m,l} + \frac{r}{2}\left(U^n_{m+1,l} + U^n_{m-1,l}\right) + \frac{r}{2}\left(U^n_{m,l+1} + U^n_{m,l-1}\right)\,. \end{aligned} \tag{30}$$

To represent this as a systems of linear equations, one arranges the $(L-1)\times(M-1)$ two-dimensional array $U_{m,l}$ into a $(L-1)(M-1)$-component vector using the lexicographical order, an example of which is shown in the lower time level of Fig. 4. The reason we called this generalization "naive" is because although it has the same accuracy and stability as the CN method in one dimension, the coefficient matrix on the l.h.s. of (30) is not tridiagonal. Therefore, it cannot be inverted time-efficiently. There are two ways out of this difficulty. One and the standard

in the community of computational scientists is to use any of Alternating Direction Implicit (ADI) methods; two well-known representatives of this family are the Peaceman–Rachford and Douglas–Rachford methods. Roughly speaking, these are special cases of operator-splitting methods, which perform the operations ∂_x^2 and ∂_y^2 in sequence. These methods are optimally time-efficient. However, both their programming and treatment of boundary conditions are nontrivial and are better be left to experts. The other way is to attempt to invert the non-tridiagoinal matrix in question using the fact that it is sparse (i.e. has nonzero entries only on 5 (not adjacent) diagonals). Matlab has special algorithms for dealing with such matrices (see the help entries for `sparse` and `spdiags`) which speed up the computations compared to the case of nonsparse matrices. The time efficiency of such an approach is sub-optimal, but it may be a worthwhile sacrifice for the ease of the programming. Alternatives to these include using `fft2` and `dct2` when boundary conditions allow, or returning to the explicit method, also sacrificing the computational speed for the simplicity of coding.

Note on hyperbolic problems

These are equations where advection strongly dominates diffusion. They occur when the unknown variable is primarily influenced by, say, a flow that carries it. They also occur in continuous age-structured models, where the role of the flow is played by the unidirectional transition from a younger subgroup to the older one. In either case, the underlying equation is (1) where $D = 0$, or its generalization for the case of two partial dimensions or several coupled variables. (Therefore, in what follows we refer only to the case $D = 0$ without mentioning this every time.) Each of such equations must have exactly one boundary condition specified at the end point where the flow emanates. For example, when solved over an interval $[0, X]$ with some $X > 0$, (1) with $g > 0$ describes advection towards increasing values of x. Therefore, such an equation must have one boundary condition at $x = 0$. For $g < 0$, the flow is in the opposite direction, and hence the boundary condition must be specified at $x = X$.

For $g > 0$, the following simple upwind method can be used:

$$\frac{U_m^{n+1} - U_m^n}{\tau} = g_m^n \frac{U_{m+1}^n - U_m^n}{h} + f_m^n, \tag{31}$$

while for $g < 0$ one needs to replace the numerator on the r.h.s with $U_m^n - U_{m-1}^n$. These methods have accuracy $O(\tau) + O(h)$. For either sign of g, a method of accuracy $O(\tau^2) + O(h^2)$ results when one uses the central difference of the form (14) for both the temporal and spatial discretizations. Note that using a wrong spatial discretization for a given temporal one (or vice versa) will result in an unstable, and therefore useless, method. The reason for this is similar to that why certain methods work well for ODEs with dissipation but do not work for conservative ODEs, and vice versa; see the companion article on ODEs for more details. Upwind, downwind, and central-difference schemes of higher accuracy can be found in textbooks. All such methods are adequate when the solution does not have a sharp propagation front. For solutions with abrupt changes either in their profile, the slope, or the curvature, it is more appropriate to use the method of characteristics. The idea of this method is founded in the fact that in hyperbolic

problems, the solutions propagate along certain lines in space and time called characteristics. Then the evolution along a characteristic is governed by an ODE rather than a PDE. The method of characteristics is easy to use either for a single equation (1) or when the coefficients g's in the system of coupled such equations do not depend on x and u's (but may depend on t).

Further Reading

Allen, L.J.S. (2007). *Mathematical biology.* Upper Saddle River, NJ: Pearson Prentice Hall.

Boyd, J.P. (2001). *Chebyshev and Fourier spectral methods* (2nd edn.). Mineola, NY: Dover.

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

Hundsdorfer, W. and Verwer, J.G. (2003). *Numerical solution of time-dependent advection-diffusion-reaction equations.* Berlin, Germany: Springer.

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.

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

Thomas, J.W. (1995). *Numerical partial differential equations: Finite difference methods*, vol. 1. New York, NY: Springer-Verlag.

See also in this volume:

Numerical methods for local models

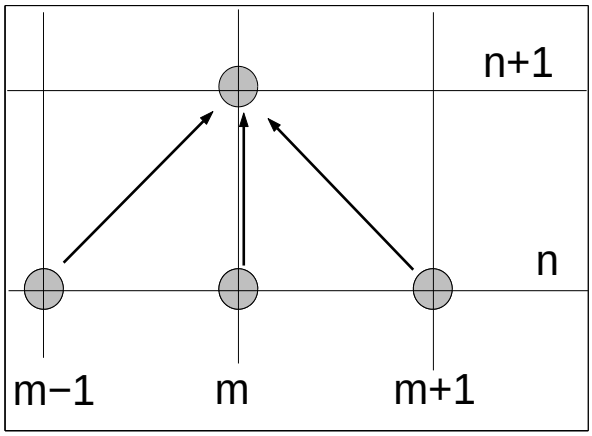

Figure 1: The stencil of method (5).

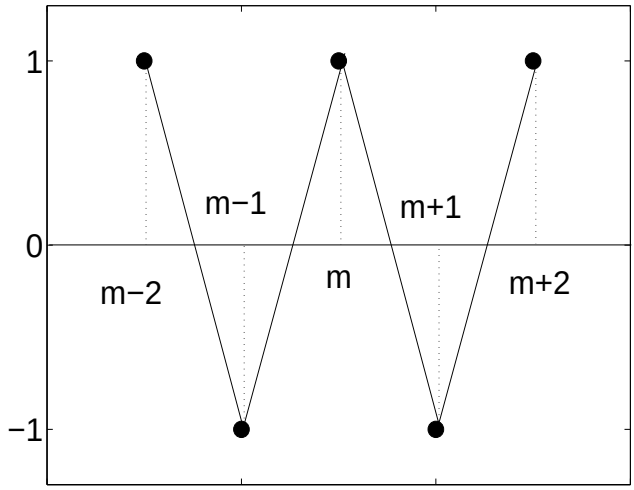

Figure 2: The schematical graph of the real part of the "fastest" spatial harmonic in (7). Its imaginary part looks qualitatively the same.

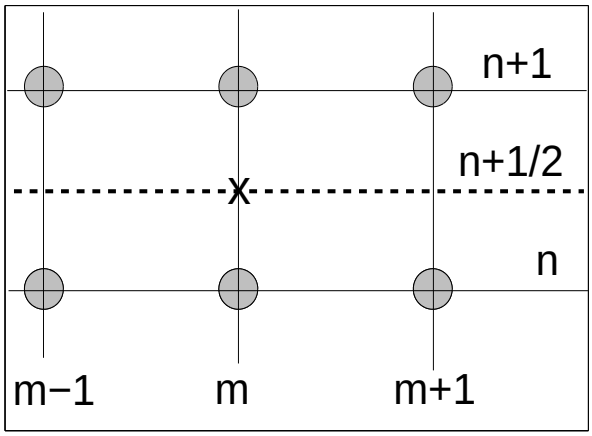

Figure 3: The stencil of the CN method (9).

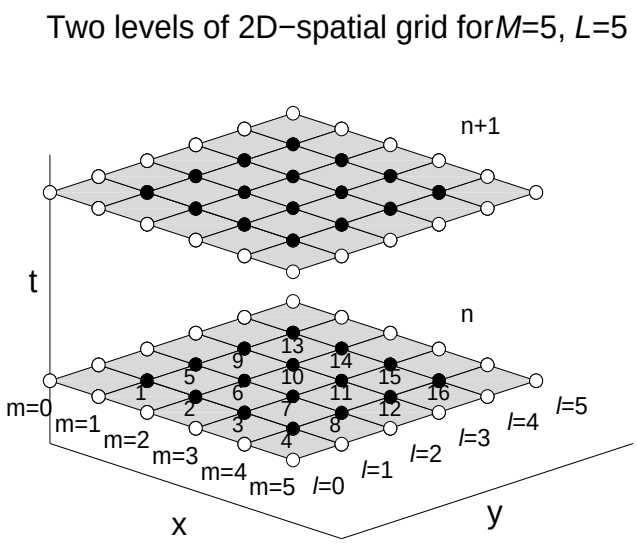

Figure 4: Schematics of the numerical grid for (26).

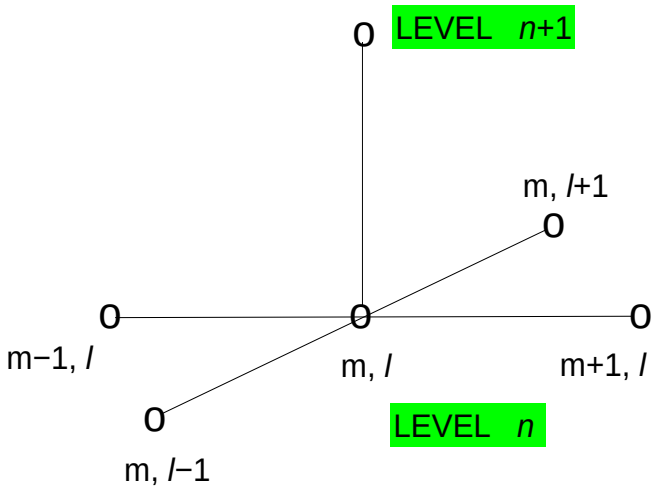

Figure 5: The stencil of method (27).