Lecture Notes and Lab Problems on Numerical Methods for Methods in Computational Neuroscience

Arthur Sherman
Mathematical Research Branch
National Institute of Diabetes and Digestive and Kidney Diseases
National Institutes of Health
Bethesda, MD 20892

Mailing address for correspondence:
Arthur Sherman
BSA Bldg., Rm. 350
National Institutes of Health
Bethesda, MD 20892
E-mail: asherman@nih.gov

August, 1997

1 Preliminaries

1.1 Formulas from Calculus

The most important formula in applied mathematics is the *Taylor series*:

$$f(x - x_0) = \sum_{i=0}^{\infty} f^i(x_0) \frac{(x - x_0)^i}{i!} \tag{1}$$

Another form:

$$f(x + h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{6} + \ldots \tag{2}$$

This can be truncated to give a representation of f in the form of a polynomial:

$$f(x + h) = f(x) + f'(x)h + f''(\xi)\frac{h^2}{2} \tag{3}$$

where $x < \xi < x+h$. If the series is truncated after one term one gets the *Mean Value Theorem* of elementary calculus. An alternative way to write this is in *Big Oh* form,

$$f(x + h) = f(x) + f'(x)h + O(h^2), \tag{4}$$

which means that the error due to truncating the series is some expression that goes to 0 as fast as h^2 as $h \to 0$. Equivalently, the error is $< Ch^2$ for some constant C when h is small.

The Taylor series is equivalent to the power series representation of functions:

$$e^x = 1 + x + x^2 + \frac{x^3}{6} + \frac{x^4}{24} + \ldots \tag{5}$$

$$\frac{1}{1 - x} = 1 + x + x^2 + x^3 + x^4 + \ldots \tag{6}$$

The latter is the standard formula for the sum of a geometric series. Comparing Eqs. 2 and 5 we get a nifty formal expression for how to advance function values from x to $x + h$ in terms of derivatives at x,

$$f(x + h) = e^{hD} f(x), \tag{7}$$

where D is the differentiation operator.

1.2 Sources of Numerical Error

A good general reference is [6]. *Precision* means how many digits can be represented in the computer. Since the number of digits is finite in practice (although some programs like *Mathematica* [17] can perform operations in arbitrary precision) irrational numbers like $\sqrt{2}$ and transcendental numbers like π and e cannot be exactly represented. Neither can repeating decimals, like 1/3 in base 10. Finally, even innocent operations between exactly represented rational numbers can lead to loss of precision. For example, in a hypothetical machine with

base 10 arithmetic and 4 decimal digits .5004e0 $\times$ .2000e1 is rounded to .1001e1, so the last digit is incorrect.

Precision can be expressed in terms of the *machine epsilon* (ϵ_{mch}), the smallest number ϵ such that $1+\epsilon > 1$. $1+\epsilon$ can $= 1$ computationally because of the need to line up the decimal (or binary) point when adding floating point numbers. Our hypothetical machine adds .1000e1 and .4999e-4 as follows:

```
.1000|0000e1
.0000|4999e1
------------
.1000e1
```

where the digits to the right of the | are lost due to rounding. Thus, $\epsilon_{mch} \approx .5e-4$ or half the last retained digit.

Typical workstations use base 2 and have a 32-bit word for single-precision real numbers, and 64 bits for double-precision. Actually, double-precision has more than twice the precision of single-precision because the number of bits devoted to the fractional part (as opposed to the exponent) is more than doubled. Typical values are 22 bits for single-precision ($\epsilon_{mch} = 2^{-23} = 1.2 \times 10^{-7}$) and 51 bits for single-precision ($\epsilon_{mch} = 2^{-52} = 2.2 \times 10^{-16}$). The rule of thumb is: always use double-precision on a 32-bit machine for numerical work. Cray supercomputers have a 64-bit word size, so single-precision is fine.

Accuracy means how close a computed answer is to the true answer. Obviously, the accuracy can be no better than the precision, but, as the above examples show, finite precision can lead to a loss of accuracy through *round-off* error. In less extreme cases, adding or multiplying two similar-sized numbers gives a relative error of $O(\epsilon_{mch})$ which is considered acceptable (because it is unavoidable).

Solving differential equations requires numerous iterated calculations, so the round-off error can accumulate. We will examine this later. Generally, however, as long as the round-off error is not amplified by the algorithm, it is not a big problem. A numerical method which does not amplify errors is called *stable*, and we will consider the stability of several algorithms later.

There is one case where round-off can be important: when two nearly equal numbers are subtracted there is a drastic loss of precision call *catastrophic cancellation*. For example, .1111 $-$ .1110 $=$ .0001. Four digits of precision are reduced to 1. This comes up in using the quadratic formula, to find a root that is near 0. For example,

$$x = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{8}$$

when c is small. This can be avoided by using the mathematically equivalent, but numerically superior, formula

$$x = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \tag{9}$$

Another example is computing e^{-5} with the power series of Eq. 5. The terms of the series alternate in sign, so massive cancellation is needed to yield a small number. A better method is to compute $1/e^5$.

The above are examples of ruining a good problem with a bad method. The fixes are problem-dependent. Sometimes, however, a problem is just intrinsically hard to solve because it is very sensitive to error (*ill-conditioned*). An example is finding the double root of x^2-4x+4. A small error $O(\epsilon)$ in computing the coefficients leads to a big error $O(\epsilon^{0.5})$ in the solution because the curve is tangent to the x-axis. Another example is trying to solve a system of linear equations whose matrix is nearly singular. This is geometrically equivalent to finding the intersection of lines (hyperplanes) that are nearly parallel. One cannot cure such cases in general; the best one can do is be aware of them.

In the case of solving $\mathbf{A}x = b$, one can estimate how ill-conditioned the problem is in the following way. With a suitably defined norm [6] $\|\cdot\|$ to measure the size of $\mathbf{A}$,

$$\|\mathbf{A}x\| \leq \|\mathbf{A}\|\|x\|$$

The natural way to test a numerical solution, $\hat{x}$, is to plug it into the equation and calculate $\hat{b} = \mathbf{A}\hat{x}$ and the *residual* $\|b - \hat{b}\|$. We will be happy if the residual is small, but what does this tell us about the error, $\|x - \hat{x}\|$? Using

$$\|x - \hat{x}\| \leq \|A^{-1}\|\|b - \hat{b}\| \qquad \text{and} \qquad \|b\| \leq \|A\|\|x\|$$

we get the estimate

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|b - \hat{b}\|}{\|b\|},$$

so we define the *condition number* of $\mathbf{A}$

$$\text{cond}(\mathbf{A}) = \|A\|\|A^{-1}\|. \tag{10}$$

If $\text{cond}(\mathbf{A})$ is large, the error may be large even though the residual is small.

A more serious way that finite resources (computing budget; disk space) and finite time (lifetime; time to get tenure; time to finish Ph. D. thesis) contribute to error is in the finite approximation of infinite limiting processes: derivatives are replaced by difference quotients; integrals are replaced by Riemann sums; and infinite sequences are truncated. This is called *truncation error* or *discretization error*. That is, the discretized problem can be solved to machine precision, but that may be only an approximation to the real continuous problem. This sort of error cannot be eliminated, but our goal is to make efficient use of the available resources. That will be the main focus below.

2 Ordinary Differential Equations

The general initial value problem we want to solve is

$$\frac{dy}{dt} = f(y,t) \tag{11}$$

with initial condition $y(0) = y_0$. This is a *first-order* differential equation. y and $f(y,t)$ can be vectors when we have a first-order system. For example, the Hodgkin-Huxley equations have $y = (V, m, h, n)$. The t dependence may reflect experimental manipulations, such as turning an applied current on and off, or other external influences, such as an imposed synaptic conductance change from another cell. We will suppress the t dependence for simplicity in many cases below.

First order systems are natural in neurobiology. If confronted with a higher order system, convert it to first order, since most solution packages assume this form. For example, the second order equation

$$z'' + 101z' + 100z = 0 \tag{12}$$

can be converted by the transformation $x = z, y = z'$ to

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -100 & -101 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \tag{13}$$

2.1 Euler's Method

The simplest method of solving ODEs is Euler's method, which directly applies Eq. 4:

$$y_{n+1} = y_n + hf(y_n). \tag{14}$$

In order to integrate from the initial data at $t = 0$ up to $t = T$, divide the interval into N equal steps of size $h = T/N$ and approximate $y(t_n = nh) = y_n$. This method works and is sometimes used in practice, but much better alternatives are described below. Nonetheless, it is the conceptual basis of all other methods, and a little analysis gives insight into how they all work.

2.2 Convergence and Accuracy of Euler's Method

It is easy to see that Euler's method converges for the special case of the equation $y' = \lambda y$ with solution $y(T) = y_0 e^{\lambda T}$. For this example,

$$y_N = y_0(1 + h\lambda)^N = y_0(1 + T\lambda/N)^N \tag{15}$$

Recalling that $\lim_{N\to\infty}(1 + 1/N)^N = e$, we see that $\lim_{N\to\infty} y_N = y_0 e^{\lambda T}$.

The following argument outlines the proof for general problems and gives an estimate of the error. You may skip to the conclusions below if you like.

From Eq. 4 we see that the error incurred in going from t_n to t_{n+1} is $O(h^2)$. Due to accumulation of these *local* errors over $N = 1/h$ steps, the global error at time T is $O(h)$. The proof follows:

Let $e_n = y(t_n) - y_n$ be the error at t_n. To see how e_n grows with n, subtract

$$y_{n+1} = y_n + hf(y_n).$$

from

$$y(t+h) = y(t) + hf(y(t)) + O(h^2)$$

to get

$$e_{n+1} = e_n + h(f(y(t)) - f(y_n)) + O(h^2)$$

Applying the Mean Value Theorem to f,

$$e_{n+1} < e_n + hMe_n + O(h^2),$$

where M is the maximum of $f(y)$ over the interval of interest. This has the form

$$e_{n+1} < \alpha e_n + \beta,$$

where $\alpha = 1 + hM$, and $\beta = Ch^2$. (FYI: This difference equation is the same one that arises in computing mortgages and annuities.) Assuming the initial error is 0, then

$$e_1 < \beta$$

$$e_2 < \alpha\beta + \beta$$

and

$$e_n < \beta(1 + \ldots + \alpha^{n-1})$$

Summing the geometric series,

$$e_n < \beta\frac{(\alpha^n - 1)}{\alpha - 1} < \beta\frac{\alpha^n}{\alpha - 1}$$

Replacing α and β by their definitions,

$$e_N < \frac{Ch}{M}(1 + hM)^N < \frac{Ch}{M}e^{hMN} = \frac{Ch}{M}e^{MT}. \tag{16}$$

Conclusions: This error estimate shows

- The global error at T is $O(h)$ (*first order accuracy*).
- The error grows exponentially in time.
- The error increases with M. This suggests that one should take smaller steps where the solution is changing more rapidly. We will return to this below.

The error analysis above ignores round-off error. If one assumes that a fixed error is added at each time step, then the error estimate of Eq. 16 is modified to $O(h) + O(\epsilon_{mch}h^{-1})$. That is, taking more steps reduces the discretization error, but increases the round-off error. Therefore, there is a point of diminishing returns where the total error increases as h decreases. Better results require not more effort, but more efficiency. The key is to take more terms of the Taylor series and reduce the discretization error to $O(h^p)$, with $p > 1$.

2.3 Higher Order Methods: Runge-Kutta

Euler's method is analogous to using a Riemann sum to evaluate an integral (quadrature) (and is in fact equivalent if the ODE is $y' = f(t)$) in that f is evaluated at only one endpoint of the time interval. By analogy with the trapezoidal rule for quadrature one can average the values of f at the left and right endpoints and obtain second-order accuracy:

$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right). \tag{17}$$

An immediate problem is that y_{n+1} appears on both the right and left sides of the equation; this is therefore said to be an *implicit* method. One approach is to use an Euler step to estimate y_{n+1}. The resulting method is called *Heun's method*:

$$z_{n+1} = y_n + hf(t_n, y_n) \tag{18}$$

$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, z_{n+1})\right).$$

This turns out also to be second order accurate, although not as stable as the genuine trapezoidal rule.

Another second order method is the *midpoint method*:

$$z_{n+1} = y_n + \frac{h}{2}f(t_n, y_n) \tag{19}$$

$$y_{n+1} = y_n + hf(t_{n+1}, z_{n+1}).$$

Both Heun and midpoint belong to the family of second order *Runge-Kutta* methods and are considered *one-step* methods since they require only the value of y at the last time step to start. Both require two function evaluations per step, *vs.* one for Euler. However, the gain in accuracy far outweighs the extra effort (see Exercise 5). Even further gains are achieved by going to the fourth order Runge-Kutta method (RK4), which samples f four times per time step:

$$m_1 = f(t_n, y_n) \tag{20}$$

$$m_2 = f(t_{n+\frac{1}{2}}, y_n + \frac{h}{2}m_1)$$

$$m_3 = f(t_{n+\frac{1}{2}}, y_n + \frac{h}{2}m_2)$$

$$m_4 = f(t_{n+1}, y_n + hm_3)$$

$$y_{n+1} = y_n + \frac{h}{6}\left(m_1 + 2m_2 + 2m_3 + m_4\right)$$

where $t_{n+\frac{1}{2}} = t_n + h/2$. Note that f is probed at each end point and twice in the middle of the interval. The analogous quadrature method is Simpson's rule, which is a weighted average of the trapezoidal and midpoint quadrature rules. This is a commonly used integration method, but not the best. For many scientists writing their own codes, it appears to represent the

psychological break-even point between investing more effort in programming *vs.* investing more CPU time.

RK4 is also a break-even point in the sense that one cannot get 5th order accuracy with 5 function evaluations; a minimum of 6 are required. Moreover, with 6 function evaluations, there are many combinations that give 4th or 5th order accuracy. This opens up a way to dramatically improve Runge-Kutta by monitoring the local error and varying the step-size. One update matches the Taylor series up to 4th order, and the other to 5th order. Thus, the difference between the two is a good approximation to the error in the 4th order update. The 5th order update is used to advance the solution, and the error estimate is used to adjust the step size. This trick and a set of usable coefficients are due to Fehlberg, but there are many variants. Implementation details and code can be found in [12]. A more naive method based on taking steps of size h and $2h$ to get an error estimate also works, but requires nearly twice as many function evaluations per step. Both simple and adaptive versions of RK4 are implemented in `dstool` and `xpp`.

2.4 Predictor-Corrector Methods

Heun's method (18) is also an example of a *predictor-corrector* method: an Euler step is used to predict y_{n+1}, and a trapezoidal rule step corrects it (improves the accuracy). To implement the full trapezoidal rule, one must solve the non-linear equation for y_{n+1}. One way is to use Newton's method. A simpler approach is to iterate the corrector step. Once an initial estimate $y_{n+1}^{(0)}$ is obtained using Euler, it can be used to get a better estimate,

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1}^{(0)})\right), \tag{21}$$

and so on until the successive values of $y_{n+1}^{(p)}$ differ by less than a prescribed tolerance. The iteration is guaranteed to converge provided that h is small enough. We will next describe a more sophisticated predictor-corrector method for ODEs, but trapezoidal rule based methods are often used for PDEs.

2.5 Multi-step Methods

Although Runge-Kutta with adaptive stepping works pretty well, for problems with complex f's, 4 function evaluations per time step may be a steep price to pay. *Multi-step* methods use the values of y at several previous time steps. The Adams methods have the general form:

$$y_{n+1} = y_n + h\int_{t_n}^{t_{n+1}} p(t)\, dt, \tag{22}$$

where $p(t)$ interpolates $f(t, y(t))$. The explicit r-step Adams-Bashforth (AB) method interpolates at $t_n, t_{n-1}, \ldots, t_{n-r}$, so

$$y_{n+1} = y_n + h\left[c_0 f_n + c_1 f_{n-1} + ... + c_{r-1} f_{n-r+1}\right], \tag{23}$$

where $f_n = f(t_n, y(t_n))$. The implicit r-step Adams-Moulton (AM) method interpolates at $t_{n+1}, t_n, \ldots, t_{n+1-r}$, so

$$y_{n+1} = y_n + h\left[\bar{c}_0 f_{n+1} + \bar{c}_1 f_n + ... + \bar{c}_{r-1} f_{n-r+2}\right]. \tag{24}$$

The r-step methods have truncation error $O(h^r)$.

AB and AM are generally used as a predictor-corrector pair. AM can be iterated repetitively until it converges, but in practice one PC step is considered best. If insufficient accuracy is obtained, it is better to reduce the time step than to iterate further.

The most commonly used members of the family are the 4-step AB:

$$y_{n+1} = y_n + \frac{h}{24}\left[55 f_n - 59 f_{n-1} + 37 f y_{n-2} - 9 f_{n-3}\right], \tag{25}$$

and 4-step AM:

$$y_{n+1} = y_n + \frac{h}{24}\left[9 f_{n+1} + 19 f_n - 5 f_{n-1} + f_{n-2}\right]. \tag{26}$$

For a derivation of the strange-looking coefficients, see [6].

The strong suit of AB/AM is that only 2 function evaluations are needed per step; each value is used 4 times, not discarded as in RK4. This is also the Achilles heel, however: to start, 4 values of y are needed. These are generally provided by RK4.

A fixed step-size version of AB/AM is included in `xpp`, but the method is most useful with variable steps. The key is that the error can be estimated from the P and C values of y_{n+1}. The local error for both AB and AM is $O(h^5)$, but with different constants:

$$y = y_P + c_P h^5 \tag{27}$$

$$y = y_C + c_C h^5 \tag{28}$$

Subtracting we have

$$0 = y_P - y_C + (c_P - c_C) h^5 \tag{29}$$

Using Eq. 28 we can eliminate h^5 to get an estimate for the error in the corrector value, y_C:

$$y - y_C \approx (y_P - y_C)\frac{c_C}{c_C - c_P}. \tag{30}$$

Using values c_C and c_P obtained from standard formulas for the error in polynomial interpolation we end up with

$$|y - y_C| \approx \frac{19}{270}\left|y_P - y_C\right|. \tag{31}$$

If y is a vector, a scaled sum of the components should be used. If the error is greater than a preset maximum, h is halved; if the error is less than the minimum, h is doubled. When h is halved, new starting values can be obtained by interpolating with RK4. When h is doubled, 4 new starting values can be obtained by using every other old step, provided 7 old points are saved. Additional implementation details can be found in [12], along with a polemic against PC methods.

2.6 Stability and Stiff Equations

So far, our choice of h has been dictated only by accuracy. We will now see that stability must also be considered. This is especially critical in problems with multiple time scales.

As we saw in Eq. 15 Euler applied to $y' = \lambda y$ gives $y_N = y_0(1 + h\lambda)^N$, which converges to $y_0 e^{\lambda t}$ as $h \to 0$. We also proved that the error decreases like h. However, if $\lambda < 0$, then not only will the error be large if h is taken too big, but the numerical solution will grow exponentially instead of decaying exponentially like the true solution. In order to guarantee a decaying solution h must satisfy

$$|1 + h\lambda| < 1 \tag{32}$$

or

$$0 < h < \frac{-2}{\lambda} \tag{33}$$

(Monotonic decay requires $h < -1/\lambda$.)

An alternative that avoids this difficulty is to use an implicit method, *backward Euler*:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}). \tag{34}$$

In general, a non-linear equation must be solved for y_{n+1}, but for our linear example we get the following recursion:

$$y_{n+1} = \frac{1}{1 - h\lambda} y_n \tag{35}$$

From the Taylor series (Eq. 5) we see that this agrees with forward Euler to first order, so it too will converge to the solution with global error $O(h)$. Furthermore, the solution will always decay for any λ which is negative. So, the solution may be inaccurate, but it will never blow up. In fact, if h is very large, the solution will be damped even more rapidly. That is, the method pushes the decaying solution prematurely towards its steady-state value of 0.

Thus, backward Euler is unconditionally stable for any equation with decaying exponential solutions, whereas forward Euler is stable conditioned on restricting h. (On the other hand, the backward Euler solution grows when $\lambda > 0$, but so does the real solution, so we can't complain.)

Applying the trapezoidal rule (17) to the same case gives

$$y_{n+1} = \frac{1 + h\lambda/2}{1 - h\lambda/2} y_n. \tag{36}$$

Like backward Euler, this decays for all $\lambda < 0$, but the multiplying factor $\to -1$ as $\lambda \to -\infty$, so the trapezoidal can suffer from slowly damped spurious oscillations.

An explicit alternative method, *exponential Euler*, sometimes used to avoid instability in linear equations of the form

$$\frac{dy}{dt} = -A(t)y + B(t) \tag{37}$$

makes the next iterate a convex combination of the current value and the steady-state:

$$y_{n+1} = y_n e^{-Ah} + \frac{B}{A}(1 - e^{-Ah}) \tag{38}$$

See [9, p. 251–254]. As $h \to 0$, this reduces to regular Euler. For large h the solution remains bounded, but it is not clear what the accuracy is. We will return to this method in the context of PDEs.

Next we give an example of a method that looks more attractive than forward Euler, but is unstable for *any* value of h:

$$y_{n+1} = y_{n-1} + 2hf(y_n) \tag{39}$$

This is an explicit, two-step, multistep method which is second order accurate (see Eq. 56); it is sometimes called the *leapfrog* method. Applying it to the equation $y' = -y$ gives

$$y_{n+1} = -2hy_n + y_{n-1}. \tag{40}$$

This is a linear second-order difference equation and can be solved analytically in a manner analogous to linear second-order differential equations [6]. Here we just state what the solution is. To make the numbers work out simply we choose $h = 3/4$. Then, if $y_0 = 1, y_1 = 0.5$, $y_n = 2^{-n}$. This exponentially decaying solution is a reasonable approximation to the ODE given the coarse time step. However, if $y_0 = 1+\epsilon$, then $y_n = (1+\frac{4\epsilon}{5})2^{-n} - \frac{\epsilon}{5}(-2)^n$. Only an $O(\epsilon)$ error is introduced into the coefficients, but over time the exponentially growing component will dominate if ϵ is not exactly 0. Making h smaller reduces the degree of instability but does not eliminate it. The decaying solution to the ODE cannot be stably computed by this algorithm. Note that forward Euler is stable for this equation with this h. Although leapfrog by itself is unstable, it is a key building block of an effective method, Bulirsch-Stoer [12].

The above examples may seem artificial. After all, the stability condition for forward Euler merely says that h can't be more than twice the time constant. To resolve behavior on the order of the time constant requires a small time step any way. If the behavior for long times is desired, it is more sensible just to look at the steady-state solutions. A real problem arises, however, when an equation has at least two disparate time constants. Eq. 12 illustrates the difficulty. The eigenvalues of the system are -100 and -1, so the general solution is

$$z(t) = C_1 e^{-100t} + C_2 e^{-t} \tag{41}$$

with the C_i determined by the initial conditions $z(0)$, $z'(0)$. The solution has a slowly decaying component and a rapidly decaying one. After a short time, the fast component is negligible, but an explicit method like forward Euler must take a time step dictated by the fast component. An implicit method will effectively take the fast process to equilibrium. That component will be solved inaccurately, but it is small, and the overall solution will be good. One can informally measure the stiffness by the ratio of the largest to the smallest eigenvalue in magnitude. Here it is 100, but in some systems, especially chemical reaction systems, the ratio can be 10^6.

Backward Euler was demonstrated here to illustrate the principle, but in practice one needs higher order methods. Forward and backward Euler can be combined in a predictor-corrector

pair. One iteration gives Heun's method, which is second order, but lacks the stability of the fully implicit trapezoidal rule. One could iterate to convergence, but convergence is slow precisely when the system is stiff. An alternative is to use Newton's method.

A family of implicit methods of order up to 5 or 6 is used in the algorithm of Gear [4]. The first order method is backward Euler,

$$y_{n+1} = y_n + hf(y_{n+1}), \tag{42}$$

The method of order k is

$$y_{n+1} - a_0 y_n - a_1 y_{n-1} - \ldots - a_{k-1} y_{n-(k-1)} = h b_k f(y_{n+1}) \tag{43}$$

Thus, like Adams-Moulton, these are implicit multistep methods, but they use old values of y rather than $f(y)$, and they evaluate f only at the right endpoint of the timestep interval. The coefficients can be derived using the elegant formalism of operator series (Ex. 23), as can the strange coefficients of the Adams family of methods [5, pp. 104–110].

In practice, to solve stiff systems it is best to use one of the many packages around that implement the Gear method [4]. `xpp` has a Gear option. If writing your own driver code, you can use the IMSL `DGEAR` subroutine or the public domain subroutine `LSODE`.

2.7 Attractors and Chaos

One way in which biology is easier than physics, is that the dynamical systems usually have stable attractors. For example, it is easier to calculate a limit cycle numerically, than the orbit of a simple harmonic oscillator, because errors in the numerical solution are damped by the trajectory's approach to a stable attractor. The error estimate for Euler's method (Eq. 16) suggests that the error grows exponentially in time. This occurs in problems with neutrally stable orbits, but not those with limit cycles (cf. Ex. 11). Similarly, when computing bifurcation diagrams, the structural stability of features such as Hopf bifurcations means they preserved in the face of small perturbations due to round-off error, although the location will be somewhat in error.

Occasionally one runs into models with chaos, which suffer from sensitive dependence on initial conditions (Ex. 12). One might think that it would be impossible to compute a meaningful solution because any error would be magnified exponentially. However, even chaotic solutions are attracted to a stable set; asymptotically the orbits are only wild within the confines of the attractor. It turns out that the numerically computed trajectory, while not faithful to the true trajectory with the given initial conditions, is shadowed by another true trajectory with different initial conditions. Thus, acceptable answers are obtained, even with simple methods like Runge-Kutta, provided we relax our standards from "Leave with the one that brought you" to "If you can't be with the one you love, love the one you're with."

Of course, there are delicate problems that are difficult to resolve numerically, but the mere existence of chaos does not necessarily invalidate numerical methods.

2.8 Choosing a Method

We close the ODE section with advice on how to choose a method for your problem. See also [12] for another point of view.

On most problems almost anything will work, even Euler if you are patient. From the point of view of efficiency, however, using adaptive methods will return the investment of programming or intellectual effort enormously.

Lower order methods, such as Heun's method, are not used much, but are included because they are closely related to PDE methods such as Crank-Nicolson that follow. There is one important exception. Heun's method is often used for stochastic differential equations where RK methods of order higher than 2 are attainable only for special systems. See [3].

Problems stiff enough to mandate using Gear are rare in neuroscience, but the `LSODE` package is a reliable, general purpose solver with good error and stepsize control and options for Adams type methods as well. There is now a version in `C`, called `CVODE`.

One problem that poses special difficulty is discontinuities. Unfortunately, these arise naturally in neuroscience when simulating voltage-clamp steps, integrate-and-fire neurons, or noise from channels or other sources. Naive implementations of multi-step methods tend to fail because they attempt to fit a smooth polynomial to past values, on the other side of the discontinuity. Robust versions of Gear like `LSODE` will isolate the singularity and restart, but it is more efficient to tell the solver about it explicitly. This is what `xpp` does. See Ex. 13. Runge-Kutta methods are simpler for problems with discontinuities because they have no memory of the past; they essentially start over at each time step. However, they will be reduced to first-order accuracy if the discontinuous events do not occur on time-step boundaries. Second-order accuracy can be achieved in integrate-and-fire networks by using linear interpolation to determine firing times between time steps [7].

There are many sources of free software, much of it of high quality. The book *Numerical Recipes* [12] is a valuable source of algorithms and codes on many topics in addition to ODEs. Consult the ODE chapter for adaptive RK, and also the Bulirsch-Stoer method which we have not covered here. On the Web, there is a large repository maintained at `netlib` (http://netlib.att.com/). The National Institute of Standards and Technology has a Guide to Available Mathematical Software (GAMS), which includes a decision tree to help locate the appropriate routine (http://gams.nist.gov). Since sites like these are subject to change and new ones are likely to emerge, look at my Web page (http://mrb.niddk.nih.gov/sherman) for updates.

The choice of method can depend in subtle ways on the nature of the problem, so it is important to be alert and flexible. It is worthwhile to experiment with more than one method to be safe. Getting answers from more than one source also helps to debug programs (or `xpp` or `dstool` input files). It is also helpful to know the answer. This is not meant facetiously: if you have knowledge of the qualitative properties of the solution or quantitative estimates of the expected sizes you will not be seduced by erroneous answers that look good. See for example Ex. 16, 17. Finally, supplementing numerical methods with analytical approaches is recommended. Programs like *Mathematica* and *Maple* help make this realistic even for non-experts.

3 Partial Differential Equations

3.1 Cable Equation

PDE's are much more complicated than ODE's, and each of the several classes requires its own solution methods. Fortunately the two main equations that come up in neuroscience are relatively easy to handle. They are the linear cable equation,

$$\tau \frac{\partial v}{\partial t} = \lambda^2 \frac{\partial^2 v}{\partial x^2} - v, \tag{44}$$

where $\tau = 1/(r_m c_m)$, $\lambda^2 = r_m/r_i$, and the nonlinear Hodgkin-Huxley equations and its variants,

$$\begin{aligned} \tau \frac{\partial v}{\partial t} &= \lambda^2 \frac{\partial^2 v}{\partial x^2} - I_{ion}(m,n,h) \\ \tau_s(v) \frac{\partial s}{\partial t} &= s_\infty(v) - s, \end{aligned} \tag{45}$$

where $s = m, n, h$ and I_{ion} is scaled by a typical conductance for comparison to the cable equation. Note that the gating variables s have no direct spatial dependence, but vary in space because v varies. The close relationship of these equations respectively to the diffusion and reaction-diffusion equations of chemistry and physics means that there is a rich legacy of techniques to draw on.

Most of the additional difficulties in going from ODEs to PDEs already arise in the linear case, so we will focus on that first.

3.2 Steady-State Cable Equation: Boundary Value Problems

For simplicity we consider first the steady-state cable equation for which time derivatives are 0 and the equation reduces to an ODE boundary value problem (BVP),

$$\lambda^2 \frac{\partial^2 v}{\partial x^2} - v = 0, \tag{46}$$

on a cable running from $x = 0$ to $x = L$. To completely specify the problem we need boundary conditions. The simplest case is to clamp v at the endpoints:

$$v(x = 0) = V_0,\ v(x = L) = V_L. \tag{47}$$

One could solve this by the *shooting* method: turn it into a system for v and v_x and solve the initial value problem with a known initial condition for v and an unknown v_x. Guess a value for v_x and integrate to $x = L$. The goal is to find what value of v_x at 0 makes $v = v_L$ at $x = L$. This is how `xpp` does it (Ex. 20).

We take a different approach which generalizes to the time-dependent case. We divide the interval $[0, L]$ at $J + 2$ points $x_j = jk, j = 0, \ldots, J + 1, k = L/(J + 1)$ (We reserve h for time step and use k for space). Note that $x_0 = 0$ and $x_{J+1} = L$. We define $v_j = v(x_j)$.

Instead of converting to a first-order system we directly discretize the second derivative of v. Using the Taylor series for v at x_j to the left and the right we have

$$\begin{aligned} v_{j+1} &= v_j + v_x(x_j)k + v_{xx}(x_j)\frac{k^2}{2} + v_{xxx}(x_j)\frac{k^3}{6} + v_{xxxx}(x_j)\frac{k^4}{24} + \dots \\ v_{j-1} &= v_j - v_x(x_j)k + v_{xx}(x_j)\frac{k^2}{2} - v_{xxx}(x_j)\frac{k^3}{6} + v_{xxxx}(x_j)\frac{k^4}{24} + \dots \end{aligned} \tag{48}$$

Adding and solving for $v_{xx}(x_j)$ we get a second-order accurate approximation:

$$v_{xx}(x_j) = \frac{v_{j-1} - 2v_j + v_{j+1}}{k^2} + O(k^2) \tag{49}$$

Thus, we replace Eq. 46 with a linear system of algebraic equations:

$$\frac{\lambda^2}{k^2}\left[v_{j-1} - 2v_j + v_{j+1}\right] - v_j = 0 \tag{50}$$

for $j = 1, \dots, N$.

If we think of each grid interval as a compartment, then Eq. 50 can be rewritten

$$\frac{1}{kr_i}\left[(v_j - v_{j+1}) - (v_{j-1} - v_j)\right] + k\frac{v_j}{r_m} = 0.$$

using $\lambda^2 = r_m/r_i$. This has the satisfying physical interpretation that, at steady-state, the sum of currents into the compartment from the neighboring compartments and the current across the membrane is 0.

Note also that by discretizing we have in a sense converted the continuous PDE into a compartmental representation of the axon. Conversely, the solution methods here apply equally to compartmental models.

We can write Eq. 50 more compactly in matrix form. Let $V = (v_1, \dots, v_J)$; we do not include v_0 or v_{J+1} because they are known and get put into the right hand side of the equation. Then

$$\mathbf{A}V = \mathbf{b}. \tag{51}$$

$\mathbf{b} = (-\frac{\lambda^2}{k^2}v_0, 0, \dots, 0, -\frac{\lambda^2}{k^2}v_L)$; $\mathbf{A}$ is a *tridiagonal* matrix:

$$\mathbf{A} = \frac{\lambda^2}{k^2}\mathbf{B} - \mathbf{I} \tag{52}$$

where I is the identity matrix and $\mathbf{B}$ is

$$\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \tag{53}$$

This matrix equation can be easily solved by Gaussian elimination in $O(J)$ time. See the Appendix for formulas and also [6, 10].

It is more common to specify the boundary conditions in terms of current instead of voltage. For example, one might inject a current I at $x = 0$ and have no current flow across $x = L$ (sealed end). The equation for axial current flow (Ohm's Law) gives

$$\frac{\partial v}{\partial x}(x = 0) = -r_i I \tag{54}$$

$$\frac{\partial v}{\partial x}(x = L) = 0. \tag{55}$$

We get a second-order accurate discretization by subtracting and solving for $v_x(x_j)$ in Eq. 48:

$$v_x(x_j) = \frac{v_{j+1} - v_{j-1}}{2k} + O(k^2). \tag{56}$$

The sealed end $x = L$ can then be satisfied by setting $v_{J+2} = v_J$. This appends an equation for v_{J+1} to Eq. 50:

$$\frac{\lambda^2}{k^2}\left[2v_J - 2v_{J+1}\right] - v_{J+1} = 0, \tag{57}$$

Similarly, we append an equation for v_0:

$$\frac{\lambda^2}{k^2}\left[2v_1 - 2v_0 + 2kr_i I\right] - v_0 = 0. \tag{58}$$

3.3 Time Dependent Cable Equation: Initial Boundary Value Problem

One approach to the time dependent case (still linear) is to view the PDE as a system of ODE's for the vector V of v_i's:

$$\tau \frac{dV}{dt} = \mathbf{A}V. \tag{59}$$

Equivalently, one can think of this as a compartmental model. One could solve this system by forward Euler,

$$V^{n+1} = V^n + \frac{h}{\tau}\mathbf{A}V^n, \tag{60}$$

at a cost of one $O(J)$ matrix-vector multiplication per time-step. (We now use superscripts to distinguish time steps from spatial grid size.) One can guess (correctly) that the accuracy of this method is $O(h) + O(k^2)$. The equations at the boundaries are modified as for the steady-state pure boundary value problem.

As we saw in the section on ODE's, Euler is stable for equations with decaying solutions (like this one) only if $h < -2/\min_m(\lambda_m)$. The eigenvalues of $\mathbf{B}$ can be computed explicitly (Ex. 19) to be

$$\lambda_m = 2\left(\cos\left(\frac{m\pi}{J+1}\right) - 1\right) = -4\sin^2\frac{m\pi}{2(J+1)}. \tag{61}$$

Then the eigenvalues of $\mathbf{A}/\tau$ are

$$\mu_m = \frac{-4\lambda^2}{k^2\tau}\sin^2\frac{m\pi}{2(J+1)} - \frac{1}{\tau}. \tag{62}$$

The first term is due to the diffusion, the second to the kinetics. If the kinetics are made slow by decreasing g, τ increases while λ^2/τ remains constant. Then, only the first term matters, and the fastest component is μ_J, for which the sin factor ≈ 1. Thus, the stability condition for forward Euler for this case is

$$h < \frac{-2}{-4\lambda^2/k^2\tau} = \frac{k^2\tau}{2\lambda^2}. \tag{63}$$

This is a **bad thing**: to achieve greater accuracy we must make k and h smaller. To maintain stability, cutting k in half requires cutting h by 4. For compartment models, the same holds as we increase the number of compartments.

Another way to view this situation is to estimate the stiffness of $\mathbf{A}$ by looking at the ratio of minimum and maximum eigenvalues ([6, 10]:

$$\frac{\mu_J}{\mu_1} = \frac{\frac{-4\lambda^2}{k^2\tau}\sin^2\frac{J\pi}{2(J+1)} - \frac{1}{\tau}}{\frac{-4\lambda^2}{k^2\tau}\sin^2\frac{1\pi}{2(J+1)} - \frac{1}{\tau}} \tag{64}$$

For large τ and large J this ratio is $O(J^2)$, i.e., the stiffness increases as the square of the number of grid points or the number of compartments. The PDE, which corresponds to the limit $J \to \infty$, can be thought of as infinitely stiff. Because $\mathbf{A}$ is symmetric the ratio μ_J/μ_1 is equivalent to the condition number of $\mathbf{A}$ (Eq. 10; see also [6]).

By doing a discrete Fourier transform of the iteration equations (60) one can see that making k smaller introduces more Fourier components with shorter wavelength into the solution [6]. Although these contribute little to the solution, they explode if the stability condition is not satisfied.

On the other hand, *increasing* g makes the cable equation *less* stiff. Then τ shrinks while λ^2/τ stays fixed and the stiffness ratio $\to 1$. Of course, the kinetics speed up, and we still need a small time step to resolve the fast kinetics, but at least the choice of h is not bound in an adverse way to the choice of k.

3.4 Implicit PDE Methods

If the problem at hand is stiff, one must resort to implicit methods. For example, backward Euler for the PDE is

$$V^{n+1} = V^n + \frac{h}{\tau}\mathbf{A}V^{n+1}. \tag{65}$$

This is linear and readily solved:

$$(\mathbf{I} - \frac{h}{\tau}\mathbf{A})V^{n+1} = V^n. \tag{66}$$

This method is unconditionally stable, like the corresponding ODE method. At each step a tridiagonal system similar to (51) must be solved. The same considerations about handling boundary equations apply. The solution at each step can be obtained in $O(J)$ operations, the same as forward Euler. The accuracy is also the same, first order in time and second order in space. In effect this means that even though stability is not compromised, h must still be proportional to k^2 for efficiency: intuitively, it makes no sense to invest an enormous effort in reducing the error due to k while the error due to h is still large (See also Ex. 22).

A better method that is second order accurate in both space and time is the trapezoidal rule (also known as *Crank-Nicolson*),

$$V^{n+1} = V^n + \frac{h}{2\tau}(\mathbf{A}V^n + \mathbf{A}V^{n+1}), \tag{67}$$

which can again be implemented by solving a tridiagonal system,

$$(\mathbf{I} - \frac{h}{2\tau}\mathbf{A})V^{n+1} = (\mathbf{I} + \frac{h}{2\tau}\mathbf{A})V^n. \tag{68}$$

(cf. Eq. 36) This takes no more work than backward Euler and is also unconditionally stable, so generally it is preferred.

Even Crank-Nicolson can not escape the clutches of the fundamental ratios τ/λ^2 (continuous) and h/k^2 (discrete). If $h/k^2 \gg 1$, $h\mu_J$ will be negative and very large in magnitude, giving rise to high frequency oscillations, sometimes called "ringing". The trapezoidal rule will damp these out, but slowly (recall Eq. 36). This comes up especially in the presence of discontinuities, as when simulating a voltage-clamp. In practice it is sufficient to take $h/k < L/\pi$, where L is the length of the cable [15, p.132]; this ensures that the high frequency components are damped more rapidly than the low frequency. Alternatively one can use backward Euler, which does not suffer from this problem, although it too will give inaccurate answers if h/k^2 is taken too large.

The program `Genesis` includes a version of exponential Euler as an alternative to implicit methods, which do not fit well with the overall structure of the package. It is applied to each line of Eq. 59,

$$\frac{dv_j}{dt} = \frac{\lambda^2}{\tau k^2}\left(v_{j-1}^n - 2v_j^n + v_{j+1}^n\right) - \frac{1}{\tau}v_j^n, \tag{69}$$

to give the iteration

$$v_j^{n+1} = v_j^n e^{-\Delta} + \Gamma(v_{j-1}^n + v_{j+1})(1 - e^{-\Delta}),$$

where

$$\Delta = \frac{2\lambda^2 h}{\tau k^2} - \frac{h}{\tau}$$

and

$$\Gamma = \frac{\lambda^2/\tau k^2}{2\lambda^2/\tau k^2 - 1/\tau}.$$

In a pure diffusion equation (no decay term in Eq. 44), $\Gamma = 1/2$. Then, for large Δ, exponential Euler sets each value to the average of its two neighbors. This will eventually converge to the

solution of the one-dimensional Laplace equation, *i.e.* the steady-state. Thus, the solution does not blow up even with large h. However, in general, it only converges to the correct solution as h, k, and $h/k^2 \to 0$, a much more restrictive condition than the stability condition for forward Euler. It may be more accurate and less expensive to use forward Euler. Exponential Euler would be at its best in non-stiff problems (few compartments/large k), with fast, linear kinetics.

3.5 Nonlinear Cable Equations

We are finally ready to solve the full non-linear Hodgkin-Huxley equations (Eq. 45) with a modified Crank-Nicolson scheme. In general, implicit methods are difficult for non-linear problems, but we can exploit a particular feature of Hodgkin-Huxley (observed by Hines [8]) to make life easier: The equation for v is linear in v if the gating variables s are held fixed, and the equations for the s are linear in s if v is held fixed. Thus, we can use the following scheme:

$$\frac{\tau}{h}\left(V^{n+1} - V^n\right) = \frac{1}{2}\frac{\lambda^2}{k^2}\left[\mathbf{B}V^{n+1} + \mathbf{B}V^n\right] + \frac{1}{2}\left[I_{ion}(S^{n+1/2}, V^{n+1}) + I_{ion}(S^{n+1/2}, V^n)\right] \quad (70)$$

$$\frac{\tau_s(V^n)}{h}\left(S^{n+1/2} - S^{n-1/2}\right) = \frac{1}{2}\left[(s_\infty(V^n) - S^{n+1/2}) + (s_\infty(V^n) - S^{n-1/2})\right] \quad (71)$$

Note the staggering of the time grids for V and S. One does a trapezoidal rule step to advance S from step $n - 1/2$ to $n + 1/2$ using V^n and does a trapezoidal rule step to advance V from n to $n + 1$ using $S^{n+1/2}$. Thereby only linear tridiagonal equations have to be solved for v at each step. (The equations for s can be solved analytically because s_i does not depend on s_j when v is fixed.) The staggering (analogous to the midpoint method) makes the process second order accurate in time.

To motivate the staggering of time consider the following simple ODE example [2, Vol. I, Chap. 9].

$$\frac{d^2x}{dt^2} = -x. \quad (72)$$

We can write this as a first order system,

$$\dot{x} = v \quad (73)$$
$$\dot{v} = -x. \quad (74)$$

This can be discretized using forward Euler as

$$x_{n+1} = x_n + hv_n \quad (75)$$
$$v_{n+1} = v_n - hx_n, \quad (76)$$

but the following alternative

$$x_{n+1} = x_n + hv_{n+1/2} \quad (77)$$
$$v_{n+1/2} = v_{n-1/2} - hx_n \quad (78)$$

is equivalent to the second order accurate discretization of Eq. 72

$$\begin{aligned} x_{n+1} - 2x_n + x_{n+1} &= (x_{n+1} - x_n) - (x_n - x_{n-1}) && (79)\\ &= h(v_{n+1/2} - v_{n-1/2}) && (80)\\ &= -h^2 x_n. && (81) \end{aligned}$$

The above trick does not work for Hodgkin-Huxley in cases where the v equation is non-linear in v. For example, one often sets $m = m_\infty(v)$ to eliminate the fast time scale of m from the problem and reduce the dimension of the system. Also, treating the driving force linearly is not always valid, especially for Ca^{2+} currents, where one might use the Goldman-Hodgkin-Katz formula. Finally, sometimes one uses polynomial-based models like Fitzhugh-Nagumo.

In those cases one must solve the nonlinear equations by iteration or by using Newton's method. Since the stiffness of these equations stems from the diffusion terms not the kinetics, iterative methods converge after a few iterations.

One scheme for iteration is:

$$\frac{\tau}{h}\left(V^{n+1,p+1} - V^n\right) = \frac{1}{2}\frac{\lambda^2}{k^2}\left[\mathbf{B}V^{n+1,p+1} + \mathbf{B}V^n\right] + \frac{1}{2}\left[I_{ion}(S^{n+1,p}, V^{n+1,p}) + I_{ion}(S^n, V^n)\right] \quad (82)$$

$$\frac{\tau_s(V^{n+1,p+1})}{h}\left(S^{n+1,p+1} - S^n\right) = \frac{1}{2}\left[(s_\infty(V^{n+1,p+1}) - S^{n+1,p+1}) + (s_\infty(V^n) - S^n)\right] \quad (83)$$

where the equations are iterated for $p = 0, 1, \ldots$ until convergence, and $V^{n+1,0} = V^n$, $S^{n+1,0} = S^n$. The linear parts of the right hand side are treated fully implicitly, while the non-linear parts are treated by predictor-corrector. One tridiagonal system must be solved for each iteration. Note that the matrix coefficients on the diagonal must be updated with the new values of the gating variables with each iteration. Often, taking one predictor step and one corrector step is adequate.

3.6 Higher Dimensions

Although cable problems are inherently one-dimensional, one sometimes needs to solve problems in two or three space dimensions. Some examples are cardiac wave propagation and Ca^{2+} diffusion in a round cell. We give just a flavor of the difficulties involved with the simple 2-D linear problem:

$$u_t = u_{xx} + u_{yy}. \quad (84)$$

For a unit square region, $0 < x < 1, 0 < y < 1$, we can let $u_{ij} = u(i\Delta x, j\Delta y), i, j = 1, \ldots, J$, or $u_{ij} = u(ik, jk)$ with $\Delta x = \Delta y = k$. Note that u_{ij} is to be interpreted as a vector of length J^2, not a matrix. Then the forward Euler discretization of Eq. 84 is

$$u_{ij}^{n+1} = u_{ij}^n + \frac{h}{k^2}\left[(u_{i+1,j}^n - 2u_{ij}^n + u_{i-1,j}^n) + (u_{i,j+1}^n - 2u_{ij}^n + u_{i,j-1}^n)\right] \quad (85)$$

or

$$U^{n+1} = \left(\mathbf{I} + \frac{h}{k^2}\mathbf{C}\right) U^n \tag{86}$$

where $\mathbf{C}$ is now an $J^2 \times J^2$ *pentadiagonal* matrix. If u_{ij} is ordered sweeping row-wise in the x-direction (like reading a page from bottom to top), $\mathbf{C}$ has a structure like

$$\left(\begin{array}{c|c|c|c|c} \mathbf{T} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \hline \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \hline \mathbf{O} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{O} \\ \hline \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{T} & \mathbf{I} \\ \hline \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{T} \end{array}\right), \tag{87}$$

where the blocks are $J \times J$, $\mathbf{I}$ is the identity matrix, $\mathbf{O}$ is the zero matrix, and $\mathbf{T}$ is a tridiagonal matrix like $\mathbf{B}$ in Eq. (53), but with -4 on the diagonal instead of -2.

The stability condition is $h < k^2/4$. The backward Euler method in the same notation is then

$$\left(\mathbf{I} - \frac{h}{k^2}\mathbf{C}\right) U^{n+1} = U^n \tag{88}$$

and Crank-Nicolson is

$$\left(\mathbf{I} - \frac{1}{2}\frac{h}{k^2}\mathbf{C}\right) U^{n+1} = \left(\mathbf{I} + \frac{1}{2}\frac{h}{k^2}\mathbf{C}\right) U^n \tag{89}$$

These are straightforward generalizations of the 1-D methods, but, unfortunately, if one attempts to solve the above matrix equations by Gaussian elimination, the zero diagonals fill in, resulting in unfeasibly large storage requirements. There is a vast literature of iterative methods for solving such matrix equations, but they are slow, especially if one has to resolve at every time step. The latter will be the case if nonlinear ionic current terms are included in the PDE because then the diagonal terms of the matrix will change at every time step. An alternative is the *Alternating Direction Implicit (ADI)* method of Peaceman and Rachford, in which one solves two consecutive tridiagonal problems, corresponding to the x and y partial derivatives. One version [13] goes as follows:

$$\left(\mathbf{I} - \frac{1}{2}\frac{h}{k^2}\delta_x^2\right) U^{n+1/2} = \left(\mathbf{I} + \frac{1}{2}\frac{h}{k^2}\delta_y^2\right) U^n \tag{90}$$

$$\left(\mathbf{I} - \frac{1}{2}\frac{h}{k^2}\delta_y^2\right) U^{n+1} = \left(\mathbf{I} + \frac{1}{2}\frac{h}{k^2}\delta_x^2\right) U^{n+1/2} \tag{91}$$

where

$$\begin{aligned} \delta_x^2 U^n &= u^n_{i+1,j} - 2u^n_{ij} + u^n_{i-1,j} \\ \delta_y^2 U^n &= u^n_{i,j+1} - 2u^n_{ij} + u^n_{i,j-1} \end{aligned}$$

In other words, $\delta_x^2 + \delta_y^2 = \mathbf{C}$. It can easily be shown (see Appendix) that this splitting of the 2-D Laplacian operator into successive 1-D Laplacians is equivalent to Crank-Nicolson to $O(h^2)$. Therefore this method has truncation error $O(h^2) + O(k^2)$ and the same stability as

Crank-Nicolson. It also has the desirable feature that in each step ($n \to n+\frac{1}{2}$; $n+\frac{1}{2} \to n+1$) J uncoupled tridiagonal systems are solved, so the method can be easily vectorized or parallelized. The method can also be extended to 3D. One source for further details is [13].

3.7 Things Left Out

We have covered how to solve space-clamped problems by solving ODEs and how to solve space and time-dependent PDEs on a single cable. Often one wants to solve on a branched structure. In an unbranched cable, each node has two neighbors giving the matrix a tri-diagonal structure. When the cable is branched, however, the nodes at the branch points have at least three neighbors, introducing far off-diagonal elements. If one is not careful about the order of the nodes, Gaussian elimination will introduce additional off-diagonal elements, complicating the solution process. Hines [8] shows how to number the branches and nodes to avoid fill-in, and also gives the formulas for spatially varying cable properties. Another approach to branching is that of Mascagni [11]. There one breaks up the structure (a neuron or a network of neurons) into sub-pieces. First the equations are solved as if the pieces were independent of each other, and then the solutions are matched at the boundaries. The virtue of this is that the sub-pieces can be solved efficiently on a vector or parallel computer. In fact, this speeds up execution so much that it would pay to take a unitary cable and artificially split it up.

4 Final Comments

We have surveyed the current wisdom on the best solutions to what might be called the easy problems. Small systems of ODE's, even stiff ones, can be solved very efficiently to high accuracy. PDE's are naturally more difficult, but reasonable methods (i.e. second-order accurate, $\mathcal{O}(\mathcal{N})$ work) are available for one-dimensional problems, including branched neurons. It is of course not difficult to come up with problems that will confound the best algorithms on the fastest computers, e.g. any problem with stiff kinetics in two or three space dimensions. We have conciously avoided venturing into these areas, both because of our own limitations and the limitations of the field as a whole.

In addition to the particular advice we have sprinkled throughout, we conclude here with some general concepts that are relevant to problems on all scales of difficulty. Numerical methods are fallible. Some may have considerable artificial intelligence built into them, but in the end there is no alternative to a deep knowledge of the particular physical problem on the part of the investigator. General dynamical systems theory can be very helpful because it categorizes possible and impossible behaviors.

There is no algorithm that solves all problems, and the user must know enough to adapt the tool to the job. It also pays to solve a problem by more than method. That means supplementing numerical methods with analytical methods and also using more than one numerical method. In addition to catching routine errors, this may uncover very subtle ones. In one small but illuminating example that we know of, an instability in the dynamical system was sensitive to numerical error introduced by the Gear method, but not Runge-Kutta, leading to discovery of a new class of phenomena (Sherman and Rinzel, 1992). No computer program can be expected to anticipate such cases. Ultimately computational science is isomorphic to all of science, and can no more than all of science ever be complete.

4.1 Appendix

4.1.1 Tridiagonal Systems

Here is the algorithm for solving tridiagonal systems from [10], reproduced for convenience. The system of equations to solve is

$$L_j V_{j-1} + D_j V_j + U_j V_{j+1} = R_j, \qquad j = 1, 2, \ldots, J$$

with $L_1 = U_J = 0$. The "forward elimination" step:

$$\begin{aligned}
U_1 &= U_1/D_1 \\
R_1 &= R_1/D_1 \\
D_j &= D_j - L_j U_{j-1}, \qquad j = 2, 3, \ldots, J \\
R_j &= (R_j - L_j R_{j-1})/D_j, \qquad j = 2, 3, \ldots, J \\
U_j &= U_j/D_j, \qquad j = 2, 3, \ldots, J-1
\end{aligned}$$

The "backward substitution" step:

$$\begin{aligned}
V_J &= R_J \\
V_j &= R_j - U_j V_{j+1}, \qquad j = J-1, J-2, \ldots, 1
\end{aligned}$$

The solution is returned in V; all the other arrays are overwritten. If this is acceptable, only 5 arrays of length J are required. The number of arithmetic operations is $O(J)$, which is optimal.

4.1.2 ADI Equivalence to Crank-Nicolson

Letting $\alpha = \frac{1}{2}\frac{h}{k^2}$, Crank-Nicolson (Eq. 89) is

$$(\mathbf{I} - \alpha\mathbf{C})\, U^{n+1} = (\mathbf{I} + \alpha\mathbf{C})\, U^n,$$

while ADI (Eq. 90–91) is

$$\begin{aligned}
\left(\mathbf{I} - \alpha\delta_x^2\right) U^{n+1/2} &= \left(\mathbf{I} + \alpha\delta_y^2\right) U^n \\
\left(\mathbf{I} - \alpha\delta_y^2\right) U^{n+1} &= \left(\mathbf{I} + \alpha\delta_x^2\right) U^{n+1/2}.
\end{aligned}$$

Combining the last two equations gives

$$\left(\mathbf{I} - \alpha\delta_y^2\right) U^{n+1} = \left(\mathbf{I} + \alpha\delta_x^2\right) \left(\mathbf{I} - \alpha\delta_x^2\right)^{-1} \left(\mathbf{I} + \alpha\delta_y^2\right) U^n.$$

Now, $\alpha\delta_y^2$ and $\alpha\delta_x^2$ are $O(h)$, so we can rewrite the above formally as

$$\left(\mathbf{I} - \alpha\delta_y^2\right) \left(\mathbf{I} - \alpha\delta_x^2\right) U^{n+1} = \left(\mathbf{I} + \alpha\delta_x^2\right) \left(\mathbf{I} + \alpha\delta_y^2\right) U^n.$$

Expanding gives

$$\left(\mathbf{I} - \alpha\delta_x^2 - \alpha\delta_y^2 + O(h^2)\right) U^{n+1} = \left(\mathbf{I} + \alpha\delta_x^2 + \alpha\delta_y^2 + O(h^2)\right) U^n,$$

which is equivalent to Crank-Nicolson up to $O(h^2)$ because $\mathbf{C} = \delta_x^2 + \delta_y^2$.

5 Exercises

Example files needed for the exercises can be found at http://mrb.niddk.nih.gov/sherman, along with updated or additional copies of these notes.

1. If the Earth (taken as a sphere with radius $r = 6378$ km) were covered with a 1 μm layer of gold, what would be the increase in surface area [16]? Compare the answers you get if you *a)* take the difference in the area before and after or *b)* use differentials to estimate the change. Which is more accurate, the exact method or the approximate method?

2. *Mathematica* knows Taylor series: Use the `Series` function to get the first 4 terms of `Exp[x]`, `Sin[x]`, and `1/(1 - x)` expanded around `0` and `f[x]` expanded around `a`. Hint: you can get the syntax by typing `?Series`. Compare the series for `Exp[I x]` and `Cos[x] + I Sin[x]` (`I = Sqrt[-1]`).

3. You can use `xpp` to test the precision of your workstation. Write a `.dif` file to calculate $(1 + (0.5)^i) - 1$. Optional: Use *Mathematica* to calculate the same function and vary the precision with the `N` command.

4. Given that the error, discretization plus round-off, of a p order method for ODEs is $h^p + \epsilon_{mch} h^{-1}$ find the minimum error attainable. What happens to the minimum h as p increases? The minimum error?

5. Consider three methods with global error $O(h), O(h^2)$, and $O(h^4)$ and requiring 1, 2, and 4 evaluations of f per time step, respectively. Calculate the total number of function evaluations needed to achieve a global error ϵ for each method.

6. Write down analytically the result of one step of RK4 applied to the equation $y' = \lambda y$ and show that the local truncation error is $O(h^5)$ by comparing with the Taylor series for $e^{\lambda h}$. *Mathematica* can help with the algebra.

7. Test the accuracy of Euler and RK4 on the equation $y' = \lambda y$ with $\lambda > 0$ using either `xpp` or `dstool`. Plot or tabulate the error at a fixed time T *vs.* h. Verify that the error increases exponentially in time with Euler.

8. Test forward Euler on $y' = \lambda y$ with $\lambda < 0$ and verify that the solution blows up if h is not small enough. `xpp` doesn't have a backward Euler menu item, but you can fake it by writing out the recursion in a .dif file and using the difference equation solver. You can compare by doing both forward Euler and backward Euler in the same file.

9. The *Mathematica* command
   `Do[{y2 = -1.5 y1 + y0; y0 = y1; y1 = y2; Print[y2]},{10}]` executes the unstable recursion of the leapfrog method (Eq. 39) 10 times. Test it with $y_0 = 1 + \epsilon, y_1 = 0.5$ for various values of ϵ including 0. (Optional: simulate leapfrog with `xpp` or `dstool` in difference equation mode and verify that it is always unstable.)

10. *Mathematica* can also calculate Eq. 39 recursively. Define `f[0] := y0 ; f[1] := y1; f[n_] := 5/2 f[n-1] - f[n-2]` and see how long it takes to evaluate `f[20]`. Show that evaluating `f[n]` takes F_n function evaluations, where F_n is the nth Fibonacci number. Can you see why recursive algorithms, while elegant, are not commonly used in numerical work?

11. Solve the predator-prey system

$$\begin{aligned} \dot{x} &= ax + bxy \\ \dot{y} &= cy + dxy \end{aligned} \tag{92}$$

with parameters $a = 0.25, b = -.01, c = -1.0, d = .01$ and initial conditions $x = 80, y = 30$. The system will oscillate with these values. Use Euler with a range of step sizes. Observe the behavior in the x-y phase plane. How small a value of Δt is needed to get good answers with Euler? RK4? After finding good numerical parameters, experiment with different initial conditions. Compare the nature of the oscillations with those of the system in the file `qbc.ode` (the model is described in [14]; you can use the default parameters in the file). Which system is more delicate to integrate numerically? Why?

12. (Taken from "Orbits Worth Betting On", Rob Knapp and Stan Wagon, CODEE Newsletter, Winter 1996.) The forced Duffing equation

$$x'' + 0.15x' - x + x^3 = 0.3cos(t)$$

is bistable between a limit cycle and a chaotic attractor. Compare the sensitivity to initial conditions when starting from $x, x' = (0.6, 1.3)$ and $x, x' = (-1.0, 1.0)$.

13. Simulate a voltage clamp with `xpp` in two ways, with a Heaviside function (using heav()) and with a global flag variable. Try both Runge-Kutta and Gear.

14. Solve the stiff system (12) with initial conditions $x(0) = 1.01, y(0) = -2$ using Euler, RK4, Adams, and Gear. How small does h have to be for each method to avoid instability? Compare the form the instability takes between methods.

15. Compare the speed of Adams and Gear on `burst.ode` with (a) $\lambda = 0.9$, (b) $\lambda = 10$, and (c) $n = n_\infty(v)$. (Case (c) will require rewriting the system.)

16. Here is an example of qualitatively wrong answers obtained because h is too big, even though the solutions look internally consistent. Integrate the file `burst.ode` with the given parameters. Establish a gold standard solution with Gear. You should see periodic bursting with 5 spikes per burst. Examining the V-S phase plane confirms the solution to be periodic. Rerun with Euler using `dt = .1, .5, 1.0`. You may also test some of the other methods to see how big `dt` can be for them.

17. This example is worse than the previous one: The problem cannot be fixed by making the step size smaller. It shows how one can be deceived by a numerical answer that

appears to be very accurate unless one knows in advance what the behavior of the system should be. Integrate the coupled oscillator system in the file qbctwo.ode. Good parameters are $s = 0.15, gc = 0.05$ and the rest at their defaults. With initial conditions `v = v2 = -47, n = n2 = .05` the cells will oscillate in phase. Now allow yourself to be led up the garden path:

- Integrate with RK with `dt = 1` for 2000 time units.
- Check your answer with Gear with the default numerical parameters. Verify that the two methods agree closely.
- Restart with Gear using the Last initial conditions for 2000 more time units. Repeat several times until you see a new pattern emerge: the cells are now oscillating anti-phase.
- Run RK for a long time. It will never produce anti-phase oscillations.
- Re-run RK, but perturb the initial condition for v2 to -47.001.

Ponder your observations and try to explain them. Which behavior is the correct one? Why do the numerical methods differ?

Moral: The computer doesn't have a brain, but you do.

18. Since a 2nd derivative is a derivative of a derivative, then it is natural to discretize it as a difference of a difference. Derive Eq. 49 by starting with Eq. 56 and using

$$v_{xx}(x_j) = \frac{v_x\left(x_{j+\frac{1}{2}}\right) - v_x\left(x_{j-\frac{1}{2}}\right)}{k}$$

19. See also [6]. Verify that the eigenvalues of $\mathbf{B}$ are given by (61) by showing that $\mathbf{B}\mathbf{u}_m = \lambda_m \mathbf{u}_m$, $m = 1, \ldots, J$ (recall that $\mathbf{B}$ is J by J) where $\mathbf{u}_m$ is the eigenvector

$$(\sin m\Delta x, \sin 2m\Delta x, \ldots, \sin Jm\Delta x),\ \Delta x = 1/(J+1).$$

Motivation: The equation $\mathbf{B}\mathbf{u_m} = \lambda\mathbf{u_m}$ is a discretization of the ODE BVP $u'' = \lambda u$ on [0,1] with u(0) = u(1) = 0; the $\mathbf{u}_m$ are the discrete representations of $\sin(mx)$ on the $[0, 1]$, which are the eigenfunctions of the ODE.

20. Solve the steady-state cable equation $v_{xx} - v = 0$ using the boundary value option in `xpp`. Let $v = 1$ at $x = 0$ and do the cases $v = 0$ and $v_x = 0$ at $x = L$ with $L = .5, 1, 2, 3$. This reproduces Fig. 21 in Rall's Handbook of Physiology article. Also do the case of a constant applied current at $x = 0$ with a sealed end at $x = 1$.

21. Solve the time-dependent cable equation $v_t = v_{xx} - v$, $v_x(0, t) = -1$, $v_x(10, t) = 0$, $v(x, 0) = 0$ using `xtc`. Write your own input file or use the file `cable.xtc`. Choose 50 grid points, so that $k = 0.2$. Test the forward Euler stability condition, Eq. 63. Observe in the unstable cases that the fastest Fourier component dominates after a few

time steps. Also try backward Euler, Runge-Kutta, and Gear. You can also use the steady-state solution obtained with the boundary value solver of `xpp` as a check on the solution at $t = 20$.

22. Minimize the error for either forward or backward Euler for PDEs for a given amount of work. Take the error to be $ah + bk^2$. The work is inversely proportional to the number of grid points and to the number of time steps, so estimate it as $\frac{C}{hk}$.

23. The backward differentiation methods (Eg. 43) can be written in terms of the backward difference operator,

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

and the differentiation operator, D. Backward Euler is then

$$\nabla y_{n+1} = hDy_{n+1}. \tag{93}$$

Rewriting Eq. 7 as

$$e^{hD} = \frac{1}{1-\nabla},$$

Eq. 93 can be expanded formally as

$$\nabla y_{n+1} = \left(\nabla + \frac{\nabla^2}{2} + \frac{\nabla^3}{3} + \ldots\right) y_{n+1}.$$

Higher-order methods are then obtained by matching more terms of the infinite series. The second-order method is

$$\left(\nabla + \frac{\nabla^2}{2}\right) y_{n+1} = hf(y_{n+1}).$$

Show that this is equivalent to

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}hf(y_{n+1}).$$

References

[1] B. Ermentrout. *PhasePlane: The Dynamical Systems Tool Version 3.0.* Brooks/Cole, Pacific Grove, California, 1990.

[2] R. Feynman. *The Feynman Lectures on Physics.* Addison-Wesley, Redwood City, CA, 1963.

[3] T. C. Gard. *Introduction to Stochastic Differential Equations.* Marcel Dekker, New York and Basel, 1988.

[4] C. W. Gear. The numerical integration of ordinary differential equations. *Math. Comp.*, 21:146–156, 1967.

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

[6] G. H. Golub and J. M. Ortega. *Scientific Computing and Differential Equations.* Academic Press, Boston, 1992.

[7] D. Hansel, G. Mato, C. Meunier, and L. Neltner. On numerical simulations of integrate-and-fire neural networks. *Neural Comput.*, in press:xx–xx, 1997.

[8] M. Hines. Efficient computation of branched nerve equations. *Int. J. Bio-Medical Computing*, 15:69–76, 1984.

[9] R. J. MacGregor. *Neural and Brain Modeling.* Academic Press, San Diego, 1987.

[10] M. V. Mascagni. Numerical methods for neuronal modeling. In C. Koch and I. Segev, editors, *Methods in Neuronal Modeling*, pages 439–484. The MIT Press, Cambridge, Massachusetts, 1989.

[11] M. V. Mascagni. A parallelizing algorithm for computing solutions to arbitrarily branched cable neuron models. *J. Neurosci. Methods*, 36:105–114, 1991.

[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Numerical Recipes.* Cambridge University Press, Cambridge, second edition, 1992.

[13] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* Interscience Publishers, New York, 1967.

[14] A. Sherman and J. Rinzel. Rhythmogenic effects of weak electrotonic coupling in neuronal models. *Proc. Natl. Acad. Sci.*, 89:2471–2474, 1992.

[15] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford University Press, Oxford, 1985.

[16] C. F. Van Loan. Using examples to build computational intuition. *SIAM News*, 28(8):1, 1995.

[17] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, Redwood City, CA, 1991.