Numerical Solution of Partial Differential Equations Finite Difference Methods

ALVIN BAYLISS

Department of Engineering Sciences and Applied Mathematics

Northwestern University, Evanston, IL 60208

1 PRELIMINARIES

In this course we will consider finite difference methods for time dependent partial differential equations (PDEs). In many instances PDEs fall into three categories. We first list these categories by giving examples from each.

- Hyperbolic equations, e.g. wave equations,

$$u_t = u_x \text{ or } u_{tt} = u_{xx},$$

- Parabolic equations, e.g., the heat equation,

$$u_t = u_{xx},$$

- Elliptic equations, e.g., Laplace equation,

$$u_{xx} + u_{yy} = 0.$$

In this course we will consider only hyperbolic and parabolic equations. These equations are supplemented by initial conditions. They are initial value problems. You specify $u(0, x)$ and solve forward in time. Thus they are model the evolution of some physical process, and time is one of the independent variables. They are also called initial value problems (IVPs) since you have to give some initial data in order to start the computation. (There is also a question of boundary conditions. This will be discussed later.)

In contrast elliptic equations are purely boundary value problems. You are given a domain together with conditions on the boundary, and have to solve in the interior. Elliptic equations are intrinsically related to linear algebra techniques.

Hyperbolic problems are characterized by waves. If we had the equation

$$u_t = u_x, \quad u(0,x) = f(x), \tag{1.1}$$

then the solution is

$$u(t,x) = f(t+x), \tag{1.2}$$

as you can easily see by just differentiating. We can interpret this as a wave traveling along the $x-$axis to the left. There is also a finite speed of propagation. Suppose for example that $f(x)$ is non-zero only in a small neighborhood of $x = 0$. Think of this as a blip or a disturbance in free space. Then for any large negative x, say $x = -a$, $u(t,x)$ will be zero until $t \simeq a$. Thus, it takes a finite time before the disturbance at $x = 0$ reaches $x = -a$. (In this case the speed of propagation of the disturbance is 1). You can also see this by recognizing that (1.1) represents an equation in the upper half of the $t-x$ plane. (Think of a Cartesian plane with the $x-$axis horizontal and the $t-$axis vertical.) Thus, $u(t,x)$ is defined for points in the upper half plane. If you now think of the line $t = -x + x_0$ in this half plane then use the chain rule of differentiation to differentiate u along this curve, (use $dt/dx = -1$ along this curve), you can see that along this curve

$$du = u_t dt + u_x dx = (-u_t + u_x)dx = 0.$$

Thus, $du = 0$ so that u does not change along this line. The equation (1.1) just propagates the initial condition $f(x_0)$ along this line. (Of course, this can also be seen from (1.2).) The line $t = -x + x_0$ is called a characteristic curve for the equation (1.1). We will discuss characteristics in more detail later.

In contrast parabolic problems are characteristic of dissipative phenomena, thus a loss of energy to friction. They are also characterized by infinite speeds of propagation. Consider for example the heat equation with trigonometric initial data,

$$u_t = u_{xx}, \quad u(0,x) = \exp(ikx).$$

We can solve this equation explicitly. To do this set

$$u(t,x) = \exp(\lambda t)\exp(ikx),$$

and plug this into the equation. We get

$$\lambda = -k^2,$$

so that

$$u(t,x) = \exp(-k^2 t)\exp(ikx), \tag{1.3}$$

and the mode decays exponentially in time with a rate that increases with k. (In contrast, if you did the same thing with the hyperbolic equation (1.1), you would get the solution

$$u(t,x) = \exp(ik(t+x)),$$

so that there is no decay in time, just a wave traveling to the left.) In the Fourier representation k is called the wave number. Note that k has units of length^{-1}. For a given wave number k, the corresponding wavelength λ satisfies $\lambda = 2\pi/k$ as you can easily see by looking at the trigonometric term $\exp(ikx)$. Note that λ has units of length.

This simple example shows that the initial data decays (dissipates) as time increases. Another property that comes from the general solution to the heat equation (which we will not give here) is that if $u(0, x)$ is non-zero only in a small neighborhood of $x = 0$, then $u(t, x)$ will generally be non-zero for all x for t arbitrarily close to $t = 0$.

There are two important points to bear in mind from these examples:

- u_x is a non-dissipative advective term characterizing the propagation of disturbances. The same thing is true for odd spatial derivatives.

- u_{xx} is a dissipative term (at least when combined with a single derivative in t). The same thing is true for even spatial derivatives, but the sign in front of the derivative matters.

  If we consider the solution to the heat equation (1.3) we see that the dissipation increases with the wave number k. Thus short wavelengths (large k) are strongly damped while long wavelengths (small k) are only weakly damped. This is a diffusion operator. Diffusion is very efficient in smoothing out small scale (large k) disturbances. It is not so efficient in smoothing out large scale (small k) disturbances. There are other ways to incorporate dissipation. Again suppose that the lefthand side is only a single derivative in time (u_t). We can include a term on the righthand side of the form $-\gamma u$ with $\gamma > 0$. It is easy to see that the solution to the equation

  $$u_t = -\gamma u,$$

  dissipates in time (the solution decays like $\exp(-\gamma t)$). The same thing is true if you had an equation like

  $$u_t = u_x - \gamma u,$$

  (do Fourier analysis - look for solutions which have a spatial dependence of the form $\exp(ikx)$. This damping is independent of k, thus it is stronger than u_{xx} on the long wavelengths and weaker on the short wavelengths. This model of damping is appropriate for some electromagnetic applications while u_{xx} is appropriate to model dissipation in a fluid.

  Dissipation can be modeled by even derivatives of any order. The term $-\gamma u$ can be thought of as a derivative of order zero. For example, you can see that a term like $-u_{xxxx}$ is also dissipative. In this case high frequencies decay like k^4 and so are really damped. This type of term is sometimes called hyperviscosity.

In summary - even spatial derivatives correspond to dissipation, but the sign in front of the derivatives is important. For the wrong sign an even derivative can correspond to explosive growth. We will see this shortly.

Solution Methods. There are 3 major solution methods for time dependent equations.

1. Finite difference methods. In these methods we break up the $x-$axis into discrete points and then approximate the solution at these points.

2. Finite element methods. In these methods solutions are approximated by a sum of functions each of which is non-zero over a small region.

3. Spectral methods. In these methods the solution is approximated by global basis functions, for example by Fourier series for periodic problems.

In this course we will consider only finite difference methods.

Hyperbolic Equations. We consider first hyperbolic equations and will consider parabolic equations later. Remember we want to generalize equations like $u_t = u_x$ which describe the propagation of waves. Ths most general case is a system

$$\vec{u}_t = A\vec{u}_x, \quad -\infty < x < \infty, \quad \vec{u}(0,x) = \vec{u}_0(x),$$

where $\vec{u}$ is an $m-$dimensional vector and A is an $m \times m$ matrix. We will define this problem as hyperbolic if A has m real distinct eigenvalues. There are other definitions of hyperbolicity in which the requirement of distinct eigenvalues can be relaxed, but we will not consider these. Note, a very simple dimensional analysis will tell you that the eigenvalues have units $length/time$, i.e., they have units of velocity. We will discuss this in more detail later.

In 2 dimensions a linear hyperbolic problem is of the form

$$\vec{u}_t = A\vec{u}_x + B\vec{u}_y,$$

where the matrix

$$\alpha A + \beta B,$$

has m real, distinct eigenvalues for all real α, β (except $\alpha = \beta = 0$).

Note that A and B can depend on x and t (and y for 2 dimensional problems). In this case the condition for hyperbolicity must hold for every x, y, and t.

There can also be a forcing term, e.g.,

$$\vec{u}_t = A\vec{u}_x + \vec{f}(x,t).$$

In this case, while the forcing term can have a dramatic effect on the solution, hyperbolicity depends only on the matrix A, i.e., only on the coefficients of the highest

order derivatives in the problem.

Examples of linear equations

1 - The one-way wave equation

$$u_t = u_x,$$

or more generally

$$u_t = cu_x. \tag{1.4}$$

Note that c has units length/time, i.e., units of velocity. $\mid c \mid$ is the speed of propagation of the waves. (Recall that velocity is a vector quantity or in 1 dimension a signed quantity - it can be positive or negative. In contrast the speed is the magnitude of the velocity and is always a positive scalar.) It is often called the sound speed since these problems arise in acoustics.

2 - The two-way wave equation written as a first order system,

$$\begin{pmatrix} u \\ v \end{pmatrix}_t = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x. \tag{1.5}$$

Note that (1.5) has solutions of the form

$$u(t,x) = v(t,x) = f(x+t), \tag{1.6}$$

and so admits waves propagating in the $-x$ direction. However, (1.5) can be reduced to the ordinary wave equation

$$u_{tt} = u_{xx}, \tag{1.7}$$

as can be seen by simply differentiating the first equation in (1.5) with respect to t and the second equation with respect to x. Since (1.7) has solutions of the form

$$u(t,x) = f(x+t) + g(x-t)$$

it (and thus (1.5)) admits solutions traveling in both the $-x$ and $+x$ direction. This is why it is sometimes called the two-way wave equation.

3 - The 2D wave equation

$$u_t = c_1 u_x + c_2 u_y. \tag{1.8}$$

Note that the speeds of propagation in the x and y directions need not be the same.

4 - The Euler equations linearized around an ambient stated of rest (governs the motion of acoustic disturbances, for example through air).

$$\begin{pmatrix} p \\ u \\ v \end{pmatrix}_t = - \begin{pmatrix} \rho_\infty c_\infty^2 (u_x + v_y) \\ p_x/\rho_\infty \\ p_y/\rho_\infty \end{pmatrix}. \tag{1.9}$$

In (1.9) the unknowns are the (generally acoustic) pressure perturbation p and the x and y components of the perturbed velocity vector (u and v). These equations are obtained by linearizing the isentropic Euler equations assuming there is no mean flow, the medium has ambient density ρ_∞ and ambient sound speed c_∞. Note that, in acoustics and other areas, ambient quantities are often denoted by the subscript ∞.

By differentiating the first equation in (1.9) with respect to t and the next two equations with respect to x and y respectively, you can see that p also satisfies the ordinary wave equation

$$p_{tt} = c_\infty^2(p_{xx} + p_{yy}). \tag{1.10}$$

Examples of nonlinear equations

Much of the phenomena of mathematical physics can not be described by linear equations. One general example of a nonlinear system of equations is

$$\vec{u}_t = A(\vec{u})\vec{u}_x, \tag{1.11}$$

where $\vec{u}$ is an $m-$vector and A is an $m \times m$ matrix. This system is hyperbolic if $A(\vec{u})$ has real, distinct eigenvalues for all $\vec{u}$ under consideration.

A very important class of nonlinear equations are systems of conservation laws

$$\vec{u}_t = \vec{f}_x, \tag{1.12}$$

where $\vec{u}$ and $\vec{f}$ are $m-$vectors. The function $\vec{f}(\vec{u})$ is called the flux function. In this case hyperbolicity depends on the Jacobian matrix

$$J(\vec{u}) = \partial\vec{f}/\partial\vec{u}.$$

By the chain rule it is easy to see that equations of the form (1.12) can be brought into the general form (1.11). In fact provided everything is sufficiently differentiable (1.12) is equivalent to

$$\vec{u}_t = J(\vec{u})\vec{u}_x. \tag{1.13}$$

Suppose for example that we are dealing with 2-vectors. We can write

$$\vec{u} = (u_1, u_2)^T, \quad \vec{f}(\vec{u}) = (f_1(u_1, u_2), f_2(u_1, u_2))^T.$$

In this case the chain rule gives

$$\vec{f_x} = \begin{pmatrix} \partial f_1/\partial u_1 & \partial f_1/\partial u_2 \\ \partial f_2/\partial u_1 & \partial f_2/\partial u_2 \end{pmatrix} \begin{pmatrix} \partial u_1/\partial x \\ \partial u_2/\partial x \end{pmatrix} = J(\vec{u})\vec{u}_x.$$

Although the equations (1.12) and (1.13) are equivalent for differentiable functions, nonlinear hyperbolic problems often have discontinuous solutions. In fact if you have ever heard of shocks in fluid dynamics, these are exactly discontinuous solutions

to nonlinear hyperbolic systems. In this case (1.12) and (1.13) are not equivalent. We say that (1.12) is in conservation form while (1.13) is in non-conservation form. One simple example is Burgers equation which in conservation form is

$$u_t = (u^2/2)_x, \quad f(u) = u^2/2.$$

The equation in non-conservation form is

$$u_t = uu_x.$$

In this course we will generally assume that all functions are sufficiently differentiable. Discontinuous solutions can be handled by some of the techniques presented here, however for strong discontinuities special numerical techniques have to be developed.

2 WELL POSEDNESS

In order to solve a problem numerically we have to be certain that the original problem is well posed. Consider the Cauchy problem

$$\vec{u}_t = A\vec{u}_x, \;\; \vec{u}(0) = \vec{g}(x), \;\; -\infty < x < \infty, \tag{2.1}$$

where A is a constant $m \times m$ matrix. The use of the term Cauchy problem means that it is a pure initial value problem. There are no boundaries. No problem can be solved on an infinite domain using finite difference techniques. There has to be boundaries somewhere. However the Cauchy problem is convenient because it is easy to analyze and to gain insight from. We will model the infinite domain by assuming periodicity over a finite domain.

The linear problem (2.1) is said to be well posed if the solutions grow no worse than exponentially. Specifically if there exists a C and α so that for all $t > 0$ we have

$$\| \vec{u}(x,t) \| \leq C \exp(\alpha t) \| \vec{u}(x,0) \| = C \exp(\alpha t) \| \vec{g}(x) \| . \tag{2.2}$$

Note that C and α have to be independent of the initial data $\vec{g}$. Also note that (2.2) implies that the solution is unique since we are talking about linear problems. In (2.2) a function space norm must be used. Typically the L_2 norm

$$\| \vec{g}(x) \|^2 = \int_{-\infty}^{\infty} \| \vec{g}(x) \|^2 \, dx,$$

is used. Note that for any x, $\| \vec{g}(x) \|$ is the usual L_2 norm from linear algebra. Note also that exponential growth of the solution is allowed for well posedness, however the growth rate must be independent of the initial data.

Example

Consider the system

$$\vec{u}_t = A\vec{u}_x, \;\; A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{2.3}$$

It is easy to see that the eigenvalues of A are $\pm i$ so that (2.3) is not hyperbolic. Suppose we consider the system in Fourier space. That is suppose our initial data is just one Fourier mode

$$\vec{u}(x,0) = \vec{g}(x) = \exp(ikx)\vec{z}.$$

The vector $\vec{z}$ is so far unspecified. Look for solutions to (2.3) of the form

$$\vec{u}(x,t) = \exp(\lambda t)\exp(ikx)\vec{z}. \tag{2.4}$$

If we substitute (2.4) into (2.3) we get an equation for λ,

$$\lambda\vec{z} = ikA\vec{z}, \tag{2.5}$$

i.e., an eigenvalue problem. Now let $\vec{z}$ be an eigenvector of A, say $x = (i,1)^T$. Then

$$A\vec{z} = -i\vec{z},$$

and we get

$$\lambda = k,, \quad \vec{u}(x,t) = \exp(kt)\exp(ikx)\vec{z}.$$

Now the wave number k can be as large as we like. Thus solutions to (2.2) grow like $\exp(kt)$ for any k. This equation is therefore not well posed.

Of course (2.2) is not hyperbolic and if the above discussion means anything we would expect hyperbolic problems to be well posed. In order to see this look at the 2-way wave equation

$$\vec{u}_t = A\vec{u}_x, \;\; A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{2.6}$$

It is easy to see that A has eigenvalues ± 1. If we do the same computation as before we now find that $\lambda = \pm ik$ depending on which eigenvector is chosen. Let us denote the two eigenvectors by $\vec{z}_+$ and $\vec{z}_-$. Thus (2.6) has solutions that look like

$$\vec{u} = \exp(ik(t+x))\vec{z_+}, \;\; \vec{u} = \exp(ik(t-x))\vec{z}_-.$$

(This should not surprise you as you know from before that solutions to the wave equations look like functions of $x+t$ and $x-t$). These solutions are bounded in time. It is easy to see that all solutions can be written as a combination of these two solutions, there is no exponential growth and (2.6) is well posed. You should also now see the importance of having real eigenvalues to the matrix A.

Constant coefficient hyperbolic problems can be shown to be well posed, It can also be shown that variable coefficient problems that are hyperbolic are well posed

(with some technical restrictions and only locally in time). For nonlinear problems there are no general results.

The Fourier analysis that we did for constant coefficient problems is often very useful in analyzing both the partial differential equations as well as the numerical approximations that we will develop.

The same analysis also works for parabolic problems. Consider

$$u_t = u_{xx}, \quad u(x,0) = \exp(ikx).$$

As we found before (see (1.3))

$$u(x,t) = \exp(-k^2 t)\exp(ikx).$$

Thus there is no exponential growth for the Fourier modes. It can be shown that this is true for general initial data and thus the heat equation is well posed. However, suppose we had

$$u_t = -u_{xx}, \quad u(x,0) = \exp(ikx).$$

We would then get as solution

$$u(x,t) = \exp(k^2 t)\exp(ikx),$$

which is terribly ill posed. Thus for even derivatives the sign in front of the spatial derivative is important. You can use this analysis to look at derivatives higher than 2. For example you can verify that

$$u_t = -u_{xxxx}$$

is well posed while

$$u_t = u_{xxxx}$$

is ill posed.

One important point for you to remember is that when the spatial derivative is even, well posedness depends on the sign in front of the derivative term. For example, the equations

$$u_t = u_{xx}, \quad u_t = -u_{xxxx},$$

are dissipative and are perfectly well posed. In contrast, the equations

$$u_t = -u_{xx}, \quad u_t = u_{xxxx},$$

are very ill posed (you can think of them as anti-dissipative). In contrast when the spatial derivative is of odd order, the sign does not matter. The equations

$$u_t = u_x, \quad u_t = -u_x,$$

are both well posed. The sign in front of the spatial derivative just determines the direction in which the waves travel. The same thing is true for odd derivatives of

higher order, e.g., u_{xxx}, as you can easily see by Fourier analysis. However, in this case the solution is not just a translation of the initial data. Different Fourier modes (i.e., different values of k) travel at different speeds. This phenomenon is called dispersion and we will discuss this in more detail later.

Finally, suppose you have several spatial derivatives of different order on the right hand side. For example, suppose you have the equation

$$u_t = u_{xx} + u_x. \tag{2.7}$$

In this case well posedness and the ultimate type of the equation is determined by the highest order spatial derivative on the right hand side. We will not give a rigorous proof of this, but it is not too difficult to see why this is true. Generally, well posedness depends on the behavior of Fourier modes for large values of k. If you consider a finite range of values of k, say $0 \leq k \leq K$, then you can generally expect to get values of C and α for the bound in (2.2) because it is over a finite range in k (at least in the context of Fourier analysis - the extension to general initial conditions is just a technical point). The problem in getting a uniform bound as in (2.2) is getting such a bound for large values of k. If you do a Fourier analysis on an equation such as (2.7) you will get

$$\lambda = -k^2 + ik, \tag{2.8}$$

and for large values of k the behavior of (2.8) is dominated by the highest power of k on the right hand side. (You can neglect the term k in comparison to k^2 for large values of k. The highest power of k corresponds to the highest order spatial derivative. Note, that this would also be true if there were a small coefficient in front of the u_{xx} term, i.e., if you considered the equation

$$u_t = \epsilon u_{xx} + u_x. \tag{2.9}$$

Equation (2.9) models a physical process with both diffusion and advection and the diffusion can have a coefficient in front of it that is as small as you like. However, when you consider large k it doesn't matter. The behavior of the Fourier representation of (2.9) is still dominated by the second derivative for large k. You can think of this as saying that when you have a combination of diffusion and advection like (2.9) the diffusion, i.e., the dissipation of the initial data, always wins no matter how small the coefficient is. It is true however that if you are not interested in small scales (k large) then many times you can ignore u_{xx} terms with a small coefficient as in (2.9). For example, the equations

$$\lambda = \epsilon k^2 + k, \quad \lambda = k,$$

give similar values of λ if you are only interested in a suitable finite range of values of k. In fluid dynamics the ϵu_{xx} term models small fluid viscosity (high Reynolds number) . For such flows you can often consider the $\epsilon = 0$ problem (inviscid flow) if you are not interested in small scales.

3 SPATIAL DIFFERENCES

Consider the equation

$$u_t = u_x, \quad u(x,0) = g(x), \quad -\infty < x < \infty. \tag{3.1}$$

Equation (3.1) is a simple, well posed hyperbolic initial value problem for $t \geq 0$. In fact you can write the solution explicitly,

$$u(x,t) = g(x+t).$$

Even though x and t enter the equation symmetrically they play different roles in the solution because you are marching in time. That is you give data at $t = 0$ and then try to determine what happens for $t > 0$. This is the essence of an initial value problem.

Before discussing numerical methods for (3.1) we consider simple ways to just approximate derivatives. Remember, you can never solve a problem on an infinite interval (at least with finite differences). In practice we will solve on a finite domain $a \leq x \leq b$ and impose boundary conditions at the boundary. We will get to this later.

Central differencing - second order

We consider the simplest way to approximate u_x by trying try to mimic what is done in basic calculus. We introduce a grid of points, $x_j = jh$ where h is the grid size (sometimes the symbol Δx is used for h). We also let u_j denote $u(x_j)$. (Sometimes u_j denotes only the computed approximation to $u(x_j)$). The simplest approximation to u_x is

$$u_x \simeq \frac{u_{j+1} - u_{j-1}}{2h} \tag{3.2}$$

.

In order to see why (3.2) makes sense we use Taylor series

$$u(x+h) = u(x) + hu_x + \frac{h^2}{2!}\, u_{xx} + \frac{h^3}{3!}\, u_{xxx} + \frac{h^4}{4!}\, u_{xxxx} + O(h^5),$$

$$u(x-h) = u(x) - hu_x + \frac{h^2}{2!}\, u_{xx} - \frac{h^3}{3!}\, u_{xxx} + \frac{h^4}{4!}\, u_{xxxx} + O(h^5).$$

After subtracting these 2 equations we get

$$u(x+h) - u(x-h) = 2hu_x + \frac{h^3}{3}\, u_{xxx} + O(h^5).$$

Simplifying we get

$$u_x = \frac{u(x+h) - u(x-h)}{2h} - \frac{h^2}{6} u_{xxx} + O(h^4). \tag{3.3}$$

If we now interpret (3.3) in terms of grid points and grid values we have

$$u_x(x_j) = \frac{u_{j+1} - u_{j-1}}{2h} - \frac{h^2}{6}\, u_{xxx}(x_j) + O(h^4). \tag{3.4}$$

You may already be familiar with the O notation. The symbol $O(h^4)$ simply means a quantity which is bounded by h^4 times a constant independent of h, i.e.,

$$| O(h^4) | \leq Ch^4.$$

The approximation

$$u_x \simeq \frac{u_{j+1} - u_{j-1}}{2h}, \tag{3.5}$$

is called a central difference approximation because it weighs each side of x_j equally. You can see from (3.4) that the leading order term of the error is $-h^2/6u_{xxx}$. This is because usually you consider h small, so that $h^2 \gg h^4$. (3.5) is called a second order approximation because the leading order term in the error is proportional to h^2. This means that if you double the grid (i.e., cut h in half) the error is reduced by a factor of 4. The leading order term of the error is sometimes called the truncation error. Note again that for all of these Taylor series type analysis you always look at the leading order term in the error (the smallest power of h that you leave out), as you assume that h is small so that for small h the lowest power of h dominates all of the higher powers.

Another way to look at this is that (3.5) is exact if $u_{xxx} = 0$ (all over not just at one point). This says that u is a quadratic, i.e., u only has powers of x up to x^2. This will generally not be the case for most computations. There will generally be an error due to the fact that u is not quadratic, and the error will be proportional to $h^2 u_{xxx}$. In applications it is difficult to quantify what this error actually is.

Fourier Analysis

A more quantitative approach to analyze errors is to do Fourier analysis. If you set $u(x) = \exp(ikx)$ then

$$u_x = iku, \tag{3.6}$$

and

$$\frac{u(x+h) - u(x-h)}{2h} = \frac{\exp(ikh) - \exp(-ikh)}{2h}\, u$$

and by simple manipulation we get

$$\frac{u(x+h) - u(x-h)}{2h} = ik\frac{\sin(kh)}{kh}\, u. \tag{3.7}$$

Equations (3.6) and (3.7) show that the effect of finite differences in Fourier space is to change the function ik (exact differentiation) to the function $ik\sin(kh)/(kh)$ (second order finite differences). Now you should know from basic calculus that

$$\lim_{z \to 0} \frac{\sin z}{z} = 1$$

Thus for fixed k we get the right answer as $h \to 0$. However, when kh is large we get the wrong answer. In fact if $kh = \pi$, $\sin(kh) = 0$ so the finite differences are completely wrong.

Points Per Wavelength

In Fourier space the error in the central difference formula $Er(k, kh)$ is

$$Er(k, kh) = ik - ik\frac{\sin(kh)}{kh} = ik(1 - \frac{\sin(kh)}{kh}). \tag{3.8}$$

The error has two components. There is a factor of ik on the outside and the relative error

$$e_2(kh) = 1 - \frac{\sin(kh)}{kh}. \tag{3.9}$$

The first term in $Er(k, kh)$, the factor ik, represents an accumulation term, i.e., errors accumulate as you compute over more and more waves. We will discuss this term later when we talk about PDEs. The relative error, $e_2(kh)$ represents the error per wavenumber k (you get it by dividing the total error Er by ik) and we discuss this error here.

What can we say about $e_2(kh)$?

1. For fixed k, $e_2(kh) \to 0$ as $h \to 0$. This is called consistency. It says that for h sufficiently small you get the right answer.

2. $e_2(kh) = O((kh)^2)$. This follows because

$$\frac{\sin z}{z} = 1 + O(z^2) \tag{3.10}$$

   for z near 0. This is just another way of saying that the difference formula is second order accurate. Note that what we are interested here is how well the function $\sin z/z$ approximates 1.

3. The relative error is only a function of kh. This is very nice for several reasons. First kh is nondimensional since k has units of length^{-1} and h has units of length. Second kh has a very nice interpretation as the number of points per wavelength. In fact if you look at the function $\exp(ikx)$ you see that

$$\exp(ik(x + \lambda)) = \exp(ikx)$$

   where

$$\lambda = \frac{2\pi}{\mid k \mid}$$

   is the wavelength of this Fourier mode. (Note that we have tacitly assumed that k is positive. This not always be the case and so we use the absolute value since the wavelength is necessarily positive.)

Now let N be the number of grid points in one wavelength. It is easy to see that $N = \lambda/h = 2\pi/\mid kh \mid$. Turning this around we have

$$\mid kh \mid = \frac{2\pi}{N}.$$

Thus the relative error for wave number k depends only on N, the number of points per wavelength. We will see that this is an important consideration in determining the error for solutions to partial differential equations but it is not the whole story because we have to consider the accumulation term, i.e., the factor ik on the outside in (3.8).

4. Unfortunately second order differencing is not very accurate. Up to leading order we have

$$\frac{\sin z}{z} = 1 - \frac{z^2}{6} + O(z^4),$$

Expressing this in terms of kh and then N we have

$$e_2(kh) \simeq \frac{\mid kh \mid^2}{6} = \frac{(2\pi)^2}{6N^2}. \tag{3.11}$$

If for example we want $e_2 \simeq 0.1$ then $N \simeq 8$. Thus to get a 10% error you need about 8 points per wavelength. We will see that in solving partial differential equations you may need much more depending on how long in time you have to solve for and how big your domain is.

Now all of these calculations are relatively straightforward. However, you might question how you get k in practice. For example, in many (most) problems you do not work with $\exp(ikx)$ but rather with Gaussians or other more general functions. There is no complete answer to this and generally this requires some intuition about the problem. What you try to do is to determine a characteristic wave length of the problem. For example, if your initial condition is a Gaussian, you might consider the Fourier transform and try to estimate a characteristic wave number k_c so that say 90% of the energy is in wave numbers below k_c.

Higher Order Difference Approximations

In many problems the second order formula is not sufficiently accurate. The computational cost of solving an equation depends on the number of grid points that you use and obviously you would like to make the number of grid points as small as possible. In order to do this we use higher order difference formulas.

The formula

$$u_x \simeq \frac{u_{j+1} - u_{j-1}}{2h},$$

approximates u_x using only the nearest neighbors of grid point x_j (i.e., x_{j+1} and x_{j-1}). It is easy to see that this formula is unique. It is the only second order formula which uses only the two neighbors. Such a formula is called compact.

If we were to use the next adjacent grid points, i.e., x_{j+2}, x_{j-2}, we could get many different formulas. For example,

$$u_x \simeq A\frac{u_{j+1} - u_{j-1}}{2h} + (1 - A)\frac{u_{j+2} - u_{j-2}}{4h},$$

is a second order approximation to u_x for any value of A.

Generally this is not a good idea. The best approximations are the compact ones. What is a good idea though is to use the additional freedom of the $x_{j\pm2}$ points to get a higher order accurate approximation. To see how this works, rewrite (3.4) using both h and $2h$

$$u_x(x_j) = \frac{u_{j+1} - u_{j-1}}{2h} - \frac{h^2}{6}u_{xxx}(x_j) + O(h^4), \tag{3.12}$$

$$u_x(x_j) = \frac{u_{j+2} - u_{j-2}}{4h} - \frac{(2h)^2}{6}u_{xxx}(x_j) + O((2h)^4), \tag{3.13}$$

Now multiply (3.12) by 4/3, multiply (3.13) by -1/3 and add. You will find that the coefficient of u_{xxx} (the leading order term of the error for the second order approximation) vanishes and we get

$$u_x(x_j) = \frac{4}{3}\,\frac{u_{j+1} - u_{j-1}}{2h} - \frac{1}{3}\,\frac{u_{j+2} - u_{j-2}}{4h} + O(h^4). \tag{3.14}$$

If we rewrite (3.14) in grid point notation and drop the error term we have the approximation

$$u_x(x_j) \simeq \frac{-(u_{j+2} - u_{j-2}) + 8(u_{j+1} - u_{j-1})}{12h}, \tag{3.15}$$

which is the unique, compact fourth order central difference approximation to u_x. By keeping track of the u_{xxxxx} term in the Taylor series you can see that the leading order term of the error is

$$\frac{-h^4\, u_{xxxxx}}{30}$$

One way to look at this is that the fourth order formula (3.14) is exact for fourth degree polynomials (quartics) where $u_{xxxxx} = 0$.

As before it is more instructive to do Fourier analysis. Let $u = \exp(ikx)$. Then apply (3.15) to u to get,

$$u_x \simeq ik(\frac{4}{3}\frac{\sin(kh)}{kh} - \frac{1}{3}\frac{\sin(2kh)}{2kh})u. \tag{3.16}$$

Observe that we do to the function $\sin(kh)/(kh)$ the same thing that we did to the Taylor series. This is true in general. We can work either with the Taylor series or the function $\sin z/z$. If we define

$$f_4(z) = \frac{4}{3}\frac{\sin z}{z} - \frac{1}{3}\frac{\sin(2z)}{2z} = \frac{8\sin z - \sin(2z)}{6z}, \tag{3.17}$$

then from what we have just done you should be able to see that $f_4(z)$ is a fourth order approximation to 1, i.e., for small z

$$f_4(z) = 1 + O(z^4),$$

(compare with (3.10)). You should now see that the relative error for fourth order differencing, $e_4(kh)$, satisfies

$$e_4(kh) = (1 - f_4(kh)), \tag{3.18}$$

(compare with (3.9). If you explicitly work out the expansions you will find that

$$f_4(kh) = 1 - \frac{z^4}{30} + O(z^6). \tag{3.19}$$

We therefore have

$$e_4(kh) = \frac{(kh)^4}{30} + O((kh)^6). \tag{3.20}$$

As before we will be primarily concerned only with the leading order term in (3.20). Equation (3.20) says that the finite difference approximation (3.15) is consistent (i.e., $e_4(kh) \to 0$ as $kh \to 0$) and fourth order accurate. If we redo the calculation that we did before to express the error in terms of points per wavelength then for a relative error of 0.1 we require

$$(kh)^4 \simeq 30 \times 0.1 = 3$$

or since $N = 2\pi/(kh)$ we have

$$N^4 \simeq \frac{(2\pi)^4}{3}, \quad N \simeq 4.8.$$

Remember that we got $N \simeq 8$ for second order differencing. Thus going to fourth order differencing reduces the resolution requirements by almost a factor of 2, just for the approximation to the derivatives. We will show that the relationship between second and fourth order differencing gets even better when you consider partial differential equations.

Finite Differences Beyond Fourth Order

You can generate approximations of any even order. For an approximation of order $2p$ we use p neighbors on each side of the point x_j (i.e., $x_{j\pm1}, \ldots, x_{j\pm p}$.) The precise weights can be derived by writing the Taylor series for $u_{j\pm1}, \ldots, u_{j\pm p}$ and taking linear combinations to eliminate all the odd derivatives of order $2l+1$ for $l = 0, \ldots, p-1$. The leading order term of the error (truncation error) is proportional to $h^{2p} d^{2p+1} u / dx^{2p+1}$.

The difference approximation can also be derived in Fourier space by taking linear combinations of

$$\sin z/z, \quad , \sin(2z)/(2z), \quad \sin(pz)/(pz),$$

so that the resulting combination equals $1 + O(z^{2p})$.

Remarks on Fourier Analysis

The first thing you might ask is how you apply Fourier analysis for typical computations of PDEs. Generally you do not work directly with $\exp(ikx)$. In most problems there is a characteristic length scale that you know beforehand. For example, you may not know exactly what the wavelength is, but you may know that the solution will oscillate on a certain scale. You can then use this scale as a characteristic wavelength, plug into the formulas and get estimates for h. In practical computations the Fourier analysis is only a guide to what your resolution requirements will be. On the other hand it is very useful because of the simplicity and quantitative nature of the formulas (and because generally you can estimate characteristic wavelengths).

We have expressed the error in terms of the nondimensional number kh, which is essentially the inverse of the number of points per wavelength. However, you should not think that kh can take all values. In fact the only relevant values for kh are

$$-\pi < kh \leq \pi.$$

In order to see why, recall that we are looking for waves of the form $\exp(ikx)$. So far k is arbitrary. But we are really looking at $\exp(ikx_j)$, i.e., we are looking at trigonometric functions on the grid $x_j = jh$. On this grid two wave numbers k_1 and k_2 give the same values if $k_1 h = k_2 h + 2\pi$. This follows from basic manipulation with complex exponentials,

$$\exp(ik_2 x_j) = \exp(ik_2 jh) = \exp((ik_2 h)j) = \exp(i(k_1 h)j).$$

Thus a high frequency (large k wave) can look like a smooth (low frequency) wave on the grid. For example. if $kh = 2\pi$ the $\exp(ikx_j) = 1 = \exp(i0x_j)$.

The phenomenon of high frequency waves looking like low frequency waves on the grid is called aliasing and can be a severe source of numerical errors.

Another way to look at this is that from the formula $kh = 2\pi/N$ we see that if $\mid kh \mid = \pi$ then $N = 2$, which is the smallest sampling you can do (it is for 2 points per wavelength). Of course, you can restrict your attention to kh in any interval of length 2π. The restriction to $-\pi < kh \leq \pi$ is natural since the formula in terms of points per wavelength works out and also because it allows a representation of the highest wave numbers which you can see on the grid.

Graphical Representation of Central Differencing

In order to pictorially see the effect of central differencing as an approximation to the operator d/dx, we can plot graphically the effect of exact differentiation and the finite difference approximation by working in Fourier space. Suppose that the x axis is now k and the limits are from $-\pi/h$ to π/h. Think of h as fixed, but for

graphical purposes it can be any positive number. We can now represent the operator d/dx in Fourier space by the line $y = k$ (we are neglecting the factor of i). Second order central differencing can be represented by the function $kf_2(k) = k\sin(kh)/(kh)$, fourth order differencing by the function kf_4 and so forth for higher order difference formulas.

What you will find is that the each curve, $y = kf_{2p}(k)$ goes through the origin (consistency). Furthermore as p increases the order of contact between the curve and the line $y = k$ (exact differentiation) at $k = 0$ increases (second order contact for f_2, fourth order contact for f_4 etc.. However, each curve moves away from the line $y = k$ as k increases. (When $kh = \pi$ all of the central difference formulas give 0). Thus higher order derivatives give you a better fit for small kh, however there are large errors as $\mid kh \mid \rightarrow \pi$. Thus finite differences are always inaccurate for such values of kh.

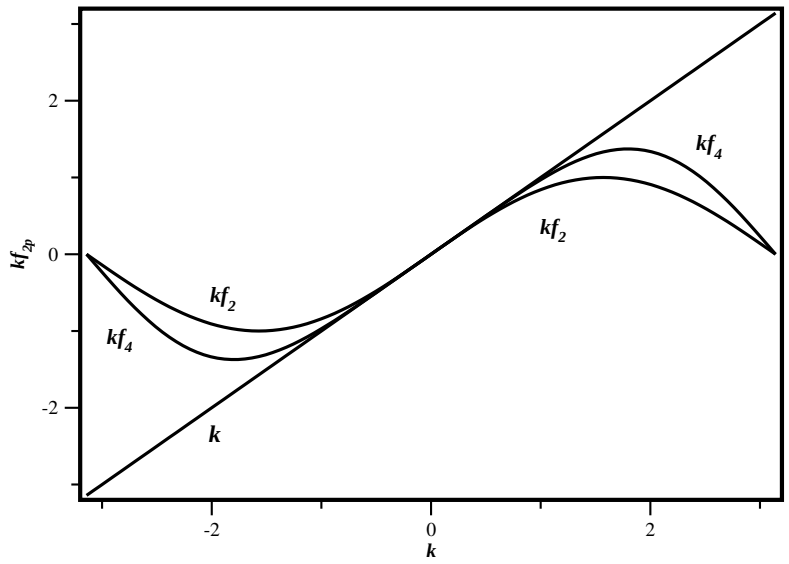

Figure 1: Graphical representation of central differencing

One-sided Differencing

Another way to approximate the first derivative is to use one-sided differences. Two such formulas are

$$u_x(x_j) \simeq (u_{j+1} - u_j)/h, \tag{3.21}$$

which is called a forward difference for obvious reasons, and

$$u_x(x_j) \simeq (u_j - u_{j-1})/h, \tag{3.22}$$

which is called a backward difference. For concreteness consider the forward difference formula. It is easy to see that this formula is first order accurate and that the leading order term in the truncation error is $h/2\, u_{xx}$. You can see this from the expansion,

$$(u_{j+1} - u_j)/h = u_x(x_j) + \frac{h u_{xx}(x_j)}{2} + O(h^2),$$

which you can derive from Taylor series.

We can also do Fourier analysis. Set $u = \exp(ikx)$ and plug into (3.21). After some manipulation you can obtain the result,

$$(u_{j+1} - u_j)/h = \frac{(\cos(kh) - 1)/h + i \sin(kh)}{h} \exp(ikx_j). \tag{3.23}$$

Recall that exact differentiation corresponds to multiplication by ik. Thus result of one sided differencing is complex rather than purely imaginary. Thus the error is no longer purely imaginary. We will not do anything more right now with central differencing, but we will see when we get to partial differential equations that the real part in the representation (3.23) can act as an additional dissipative term and damp out oscillations in the numerical approximation/

Approximations of Second Derivatives

If we have a second order equation, for example

$$u_{tt} = u_{xx}, \text{ or } u_t = u_{xx},$$

then you have to compute an approximation to u_{xx}. Manipulation of the Taylor series for u around x_j gives,

$$u_{xx}(x_j) = \frac{u_{j+1} + u_{j-1} - 2u_j}{h^2} - h^2 \frac{u_{xxxx}}{12} + O(h^4). \tag{3.24}$$

You can see from (3.24) that the second order, central difference approximation to u_{xx} is

$$u_{xx}(x_j) \simeq \frac{u_{j+1} + u_{j-1} - 2u_j}{h^2},$$

and the truncation error is

$$-h^2 \frac{u_{xxxx}}{12}.$$

Note that the coefficient of the truncation error (1/12) is smaller than for the first derivative (1/6) (see 3.4). This is because you are being more compact. You are using the nearest neighbors to get a second derivative rather than first derivative.

The Fourier analysis of (3.24) is very similar to what we did for the first derivative. Set $u(x) = \exp(ikx)$, apply (3.24) and compare to the result of exact differentiation. We get

$$u_{xx} = -k^2 u, \;\; u_{xx} \simeq -2k^2 \frac{1 - \cos(kh)}{(kh)^2}.$$

The relative error (dividing the error by k^2) is now

$$E_2(kh) = \mid 1 - 2(\frac{1-\cos(kh)}{(kh)^2}) \mid .$$

As we did before we can expand for small kh to get

$$E_2(kh) \simeq \frac{(kh)^2}{12}.$$

You should compare this with the result for the first derivative (3.11) where the relative error was $(kh)^2/6$. Again for a give kh the relative error is smaller for approximation to the second derivative than for the first derivative.

4 TEMPORAL ERRORS

So far we have not considered the effect of advancing the solution in time. We will now consider what happens when we approximate partial differential equations. We will begin with the simple initial value problem

$$u_t = u_x, \;\; u(x,0) = g(x), \;\; -\infty < x < \infty. \tag{4.1}$$

This problem is very simple. You can write down the solution,

$$u(x,t) = g(x+t).$$

You might ask why we consider such a simple problem. It is typical that when dealing with partial differential equations and their numerical solution general, rigorous results are not available. For example for many simple looking nonlinear partial differential equations there are no proofs of existence. Thus for complicated equations it is almost impossible to study and prove anything about the properties of numerical approximations. The idea is to study the properties of numerical approximations for simple problems such as (4.1) to get insight as to how the methods behave for more complicated problems.

By now we know how to approximate the u_x. Let's introduce some simpler notation. Set

$$D_2(h)u = \frac{u(x+h) - u(x-h)}{2h}.$$

It is simplest to think of $D_2(h)$ as mapping functions in L_2 into other functions. In other words think of $D_2(h)$ as operating on functions rather than on sequences. We will omit the dependence of D_2 on h unless there is a possibility of confusion.

One way to approximate the solution to (4.1) is to solve the equation

$$v_t = D_2 v, \;\; v(x,0) = g(x), \;\; -\infty < x < \infty, \tag{4.2}$$

(note v denotes the approximate solution, it is not equal to u) by approximating v_t by some of the methods we have learned previously.

What can happen? Basically two things:

1. The resulting approximation can be unstable. When this happens the numerical solution can grow exponentially and even faster. (Note that there is no growth associated with the underlying equation (4.1)). This is analogous to an ill posed partial differential equation.

2. The resulting approximation can be perfectly stable but inaccurate. To some extent this is **more** pernicious than instability because if your approximation is unstable you will know it. The numbers will overflow in the computer. On the other hand, if you are getting stable numbers you may be fooled into thinking that the answer is accurate when it could be very inaccurate. (Remember that in general you will only be solving partial differential equations numerically when you do not know the answer beforehand).

Continuous in Time Approximation

Before discussing actual methods, i.e., discrete methods in space and time, we will discuss one more approximation, namely semi-discrete approximations. In particular consider the system (4.2) and let us consider the exact solution to this equation. Thus we are discretizing in space but not in time. This significantly simplifies the analysis. Remember that you should think of the operator $D_2(h)$ as a mapping of the function space L_2 into itself.

We are thus solving the semi-discrete approximation (4.2). Of course, in reality you cannot do this. You have to introduce some discretization in time. However, the semi-discrete approximation can give you a lot of insight into the nature of the numerical errors. It will turn out that for central differencing (not necessarily for one-sided differencing) the semi-discrete problem is stable in time, i.e., solutions do not blow up. Thus, we only have to worry about the causes of inaccuracies (point 2 above). These numerical errors fall into two general categories:

1. Dispersion

2. Dissipation

Dispersion occurs when for the computed solution different wave numbers have different phase velocities. (We will define what this means shortly).

Dissipation is where there is a spurious loss of energy due to the numerical approximation.

We first discuss dispersion and digress to describe two velocities associated with wave propagation. Consider a linear system with constant coefficients. Suppose that

for each wave number k we have solutions of the form

$$\vec{u}(x,t) = \exp(i\omega(k)t)\exp(ikx)\vec{z}, \tag{4.3}$$

for some vector $\vec{z}$. Clearly this describes a wave where for each wave number k the frequency is $\omega(k)$. The quantity $d\omega/dk$ has units of length/time and is thus a velocity. (Note that ω itself has units of $1/t$). In addition, the quantity ω/k also has units of length/time and is a velocity associated with the wave. What is the difference between these two velocities? The phase velocity of the wave with wavenumber k is $-\omega/k$. This simply says that if you consider a wave of the form (4.3) and consider a curve $x(t)$ in the $x-t$ plane where the phase is constant, you will find that $dx/dt = -\omega/k$ (just set the time derivative of the phase equal to 0). In contrast, if you look for a curve in the $x-t$ plane where the phase is stationary (i.e., the derivative of the phase with respect to k is zero), you will find $x/t = -d\omega/dk$. The quantitiy $-d\omega/dk$ is sometimes called the group velocity and describes the rate at which energy propagates in a packet of waves.

We will not pursue this further, as an analysis of such energy flow is not important for the development of numerical methods. However, we will discuss the ramifications of dispersion since such dispersion will be introduced by numerical discretizations. A system is said to be non-dispersive if

$$d\omega/dk = \text{constant}.$$

Otherwise a system is dispersive.

So far the discussion of dispersion has been more related to wave propagation than to numerics. However, consider the example,

$$u_t = u_x.$$

Simple algebra (try a solution of the form $\exp(i\omega t)\exp(ikx)$) gives the dispersion relation

$$\omega = k.$$

Thus this equation is non-dispersive. To see a dispersive system consider the equation

$$u_t = u_x + u_{xxx}. \tag{4.4}$$

The dispersion relation is

$$\omega = k - k^3.$$

Thus $d\omega/dk$ is not constant and this system is dispersive. In generally dispersive systems are really exciting because the fact that different wave numbers have different velocities means that a simple pulse can have very complex behavior as it evolves in time since different Fourier modes propagate with different velocities. For example, if you start with a simple, localized pulse (for example a Gaussian function of x), a long oscillatory tail can develop. On the other hand if you had a non-dispersive system then the pulse propagates essentially unchanged.

Unfortunately

1. Many interesting physical problems are non-dispersive, at least to a very good approximation.

2. Numerical approximations introduce spurious numerical dispersion so that very interesting, but wrong, solutions are computed

3. It is very common for people not familiar with numerical dispersion to "discover" interesting results, which are not real but rather arise from spurious numerical dispersion.

Analysis of Numerical Dispersion

Consider the equation

$$u_t = u_x \tag{4.5}$$

and the semi-discrete approximation

$$v_t = D_2(h)v. \tag{4.6}$$

Suppose for both equations we have the initial condition

$$u(x,0) = v(x,0) = \exp(ikx).$$

We can now write down the exact solutions in the form

$$u(x,t) = a(t)\exp(ikx), \quad v(x,t) = b(t)\exp(ikx) \tag{4.7}$$

(Fourier analysis again). Plugging into the equations we can derive ordinary differential equations for a and b. We get

$$\dot{a} = ika, \quad a(0) = 1, \tag{4.8}$$

and

$$\dot{b} = ik\frac{\sin kh}{kh}b, \quad b(0) = 1. \tag{4.9}$$

Note that (4.9) follows directly from the central difference approximation in Fourier space (see (3.7)). If we solve for a and b we get

$$a(t) = \exp(ikt) = \exp(i\omega_u t), \tag{4.10}$$

and

$$b = \exp(ik\frac{\sin kh}{kh}t) = \exp(i\omega_v t). \tag{4.11}$$

If you recall the form of the exact solutions (4.7), you can see that the PDE solution is non-dispersive ($d\omega_u/dk = 1$) while the numerical approximation is dispersive because ω_v depends nonlinearly on k.

This should not surprise you. Remember from (3.4) that accounting for the leading order term in the truncation error we have

$$D_2(h) = (v(x+h) - v(x-h))/(2h) = v_x + h^2 v_{xxx}/6 + O(h^4).$$

Thus v in (4.6) really solves the equation

$$v_t = v_x + h^2 v_{xxx}/6 + O(h^4), \tag{4.12}$$

and we saw that adding a third derivative is dispersive (see (4.4)). Equation (4.12) without the $O(h^4)$ term is sometimes called the modified equation. Up to leading order it is the PDE that is solved by the numerical approximation.

Using these formulas we can see how many points per wavelength we really need in solving PDEs. Compare the exact solution,

$$u(x,t) = \exp(ik(x+t))$$

with the approximate solution

$$v(x,t) = \exp(ik(\frac{\sin\, kh}{kh}x + t)).$$

(You might ask where did the ω's go? We have assumed the speed of propagation in the original equation is unity, i.e., we work with the equation $u_t = u_x$, instead of the equation $u_t = cu_x$ where c is the speed of propagation, for example the sound speed if you are working in acoustics. In this case both t and x are nondimensional or equivalently have the same units so that for the exact solution $\omega = k$ rather than $\omega = ck$. Nondimensionalizing so that we can take $c = 1$ simplifies the formulas but doesn't really change the analysis or the results.) You should now see that the only errors are in the phase and are due to dispersion.

Slow down for a minute and reflect on what we have done. Previously we looked at errors in approximating the derivative in Fourier space. The effect in Fourier space is to replace exact differentiation (multiplication by ik) by multiplication by $ik\sin(kh)/(kh)$. However, we did not discuss how this error manifests itself. Now we looked at a PDE using the semi-discrete approximation. We showed that the errors in the spatial derivatives are manifested in the phase (i.e., in the time dependence of the wave). Furthermore, we previously considered only the relative error by not considering the ik factor in front of the $\sin(kh)/(kh)$ term. Now that we see the total error is in the phase and is multiplied by t, we have to consider what happens as t increases. We will see that the factor ik in front of $\sin(kh)/(kh)$ has the effect of increasing the error as t increases. This is because phase errors will accumulate over many wavelengths.

To see this, let P_u and P_v be the phases for u and v. We have

$$P_u = k(x+t), \quad P_v = k(\frac{\sin kh}{kh}t + x). \tag{4.13}$$

You can see from (4.13) that the phase error grows linearly with t. Thus for any given kh the solution gets worse the longer you integrate for. Said another way, in order to control the phase error the number of points per wavelength depends on how long you integrate for, i.e., how long you want to follow the solution. This is a manifestation of the accumulation of phase errors.

At time t the phase error $e(kt, kh)$ depends on kt and kh. It is given by

$$e(kt, kh) =\mid P_u - P_v \mid=\mid kt(1 - \frac{\sin kh}{kh}) \mid\simeq\mid \frac{kt(kh)^2}{6} \mid . \tag{4.14}$$

Now let's normalize by the number of periods in time that you want to compute for. It is easy to see that for a given k, the solution is periodic in t with period $T = 2\pi/k$. Suppose that we want to follow the solution for J periods so that $t = JT$. We have

$$t = JT = \frac{J2\pi}{k} \rightarrow kt = J2\pi.$$

If we plug this value of kt into (4.14) and treat the phase error e as a function of J and kh, we get

$$e(J, kh) \simeq\mid \frac{J2\pi(kh)^2}{6} \mid .$$

Now remember that kh can be related to N, the number of points per wavelength ($N = 2\pi/(kh)$. We can then replace treat e as a function of J and N and get

$$e(J, N) \simeq\mid \frac{J(2\pi)^3}{6N^2} \mid . \tag{4.15}$$

Note that in applications J and N are more physically relevant than kt and kh. Remember that the model problem (4.5) and the Fourier analysis are just a guide for what to expect in more complicated problems. Usually you will have some idea of a characteristic wavelength for the problem and some idea for the characteristic period and the number of periods that you want to solve for.

Finally we can turn (4.15) around to get a formula for N in terms of J, for a given error level e,

$$N \simeq \sqrt{J(2\pi)^3}/\sqrt{6e}. \tag{4.16}$$

You can now estimate N (the required resolution) given J (length of time you want to solve for) and an error level e. Suppose for example $e = 0.1$. We then have

$$N \simeq 20\sqrt{J}.$$

If we want $e = 0.01$ we have

$$N \simeq 64\sqrt{J}.$$

Let us summarize this. In general you want the phase errors to be constant over the whole computation. However, the phase errors will accumulate as you solve the

equation. Thus you have to account for J, the length of time of the computation in assessing resolution requirements. For a given error level the number of points per wavelength increases as $\sqrt{J}$ (i.e., as $\sqrt{t}$). For example, even for the crude error $e = 0.1$, if $J = 9$ then $N \simeq 60$. Since if you integrate for a longer time, the waves travel a greater distance you can also interpret this as saying that N increases with the size of the spatial region of your computation (the number of waves you want to compute).

You should easily be able to repeat the same analysis for fourth order differences. Instead of (4.14) the phase error is now

$$e = kt(1 - f_4(kh)),$$

where $f_4(z)$ is defined in (3.17). Since

$$1 - f_4(kh) \simeq \frac{(kh)^4}{30},$$

for small values of kh (see (3.20) we get an expansion for the error,

$$e \simeq \mid \frac{J(2\pi)^5}{30N^4} \mid \tag{4.17}$$

where as before J is the number of periods and N is the number of points per wavelength. We can now rewrite (4.17) to express N in terms of e and J,

$$N \simeq J^{1/4}(2\pi)^{5/4}/(30e)^{1/4}. \tag{4.18}$$

Thus as you increase the number of periods, N increases as $J^{1/4}$ for fourth order differencing as opposed to $J^{1/2}$ for second order differencing. You should know that the function $J^{1/4}$ increases much more slowly as $J \to \infty$ than the function $J^{1/2}$. If, for example, we set $e = 0.1$ we get

$$N \simeq 7.5J^{1/4},$$

so that if $J = 9$, then $N \simeq 13$ (as opposed to about 60 for second order differencing).

We should point out that these estimates are for the semi-discrete problems. Differencing in time can sometimes improve the accuracy because of a cancelation of the phase errors, however the scaling of N with J will generally be unchanged.

You can extend this to central differencing of order p, where p is even. You will find a general scaling relationship of the form

$$N \simeq J^{1/p}$$

for p^{th} central differencing.

Numerical Dissipation

The above analysis showed that central differencing was non-dissipative for the semi-discrete problem. That is we started with $\exp(ikx)$ and got back a dispersive solution but with the same amplitude. Another way to look at this is to see that when we did the Fourier analysis we got a real ω.

To see the effect of numerical dissipation let's consider, instead of central differencing (4.6), the approximation

$$v_t = D_+ v, \tag{4.19}$$

where D_+ is the forward difference operator (see (3.21)),

$$D_+ v = \frac{v(x+h) - v(x)}{h}. \tag{4.20}$$

Now (4.20) is only first order accurate so it is not clear why you would want to do this. Let's do the Fourier analysis again. Set $v(t) = c(t)\exp(ikx)$ and plug in to (4.19) to get

$$\dot{c} = (\frac{-1 + \cos kh}{h} + i\frac{\sin kh}{h})c,$$

which we can write as

$$\dot{c} = ik\frac{\sin kh}{kh}c - \frac{1 - \cos kh}{h}c. \tag{4.21}$$

Now if we write $c(t) = \exp(i\omega(k)t)$ we get

$$i\omega = ik\frac{\sin kh}{kh} - \frac{1 - \cos kh}{h}, \tag{4.22}$$

and we see that ω is no longer real. Now, since the real part of $i\omega$ is less than 0, $(1 - \cos kh > 0)$, $c(t)$, and hence the solution to (4.20), decays as t increases. It is important for you to realize that this decay is purely due to the numerical discretization. There is no decay associated with the underlying partial differential equation. We can go a little further. Suppose kh is small and use the Taylor expansion for the real part in (4.22)to get

$$\frac{1 - \cos kh}{h} \simeq \frac{k^2 h}{2},$$

i.e., the first order accuracy manifests itself in a decay rate which is $O(h)$. (Recall that units of time and space are indistinguishable for (4.1) since we take the speed of sound =1.)

This should raise a question in your minds. We saw that if we did a forward difference the solution would decay in time. Without doing the algebra you should expect that doing a backward differencing (3.22) should switch the signs so that the solution should grow exponentially in time. In fact it does. This is a manifestation of numerical instability. A seemingly small change in the numerical procedure can give

you explosive growth for a problem for which the analytic solution does not grow. You will see many other examples of this.

You can ask why do forward and backward differencing have such dramatically different effects. Look at the basic equation (4.1). The general solution is $u = g(x+t)$, where $g(x)$ is the initial data. This describes a wave moving to the left. For example if g is a pulse centered at $x = 0$ then for large times $g(x + t)$ is a pulse centered at $x = -t$. Another way to look at this is that the solution is constant along the family of curves $t + x =$ constant. These are the characteristic curves for (4.1). The characteristic curves travel to the left as t increases. You can think of the initial data being propagated along the characteristic curve, for example the value of $g(0)$ the initial data at $x = 0$ is carried to the left on the characteristic curve $x + t = 0$. Consider a fixed value of x, say $x = x_1$. At any given time $u(x_1, t)$ depends on values of u at earlier values of t but also on values of x to the right of x_1. On the other hand the solution at this time does not depend on values of x to the left of x_1. It is therefore natural to use the forward difference and the analysis confirms that this is stable and indeed dissipative. This is called upwind differencing. It is not natural to use the backward difference and the analysis confirms this by showing the backward differencing (downwind differencing) leads to numerical instability. Often (but not always) numerical instability results from doing something in your approximation that violates the physical principles underlying the equation and the solution.

The last question that you might ask is why you might want to use a method which gives decay to the solution. To understand why, suppose $kh = \pi$ so that you have only 2 points in the wave. On your grid this mode looks like +1,-1,+1,-1. It is sometimes called the $2\Delta x$ mode because it is periodic with a period $2\Delta x = 2h$. Using central differencing any such component in the solution will persist because $\sin(kh) = 0$ for $kh = \pi$. Using upwinding however, this mode will decay in time. Thus upwinding is good to get rid of spurious high frequency waves (i.e., waves that oscillate on the scale of the grid) that central differencing can not damp. Upwinding is also good for complex nonlinear problems which are prone to numerical instabilities that you can not readily analyze.

5 MOSTLY EXPLICIT DIFFERENCE SCHEMES

Consider a general hyperbolic system,

$$\vec{u}_t = A(x,t)\vec{u}_x, \quad -\infty < x, \infty, \quad \vec{u}(0,x) = \vec{g}(x), \tag{5.1}$$

where for every x and t, $A(x,t)$ is a real $m \times m$ matrix with distinct real eigenvalues so that (5.1) is hyperbolic. When (5.1) is discretized in both space and time we obtain a difference scheme. Both the theory and notation for difference schemes are much more complicated than for the semi-discrete approximation that we considered

previously. Because of this we will do as much as we can by simple examples. In particular, we will consider the basic initial value problem equation for the one-way wave equation,

$$u_t = u_x, \;\; u(x,0) = g(x), \;\; -\infty < x < \infty. \tag{5.2}$$

In many instances it is straightforward to generalize methods for (5.2) to the more general system (5.1).

We next discuss some basic notation. We will use h and Δx interchangeably to denote the grid spacing in x. We will generally use Δt to denote the timestep although occasionally we may use k if it makes the formulas simpler. For an approximation to the solution at time $t_n = n\Delta t$ and at $x_j = jh$ we will use the notation v_j^n. Thus v_j^n is an approximation to $u(x_j, t_n) = u_j^n$. We will also use the notation $\lambda = \Delta t/h = \Delta t/\Delta x$.

Forward Euler

In many difference schemes you decouple differencing in time from differencing in space. That is you use some differencing in x, get the semi-discrete approximation and then use some differencing in t to get your difference scheme. The simplest case is when you use central differencing in x and forward Euler in t. The scheme is

$$v_j^{n+1} = v_j^n + \lambda/2 \; (v_{j+1}^n - v_{j-1}^n) \tag{5.3}$$

Before we analyze (5.3) we make two observations. First while we expect Δt and h to be small it does not follow that λ is small. In fact the size of λ will be very important in analyzing difference schemes. Second, you should see the general form for λ. If instead of (5.3) we had the more general equation

$$u_t = cu_x,$$

where c is a velocity (think of it as the sound speed) then you would get the same formula as in (5.3) provided λ is defined as

$$\lambda = c\Delta t/\Delta x. \tag{5.4}$$

Thus everything is the same as for $c = 1$, except that λ has to be replaced by (5.4). This will be true for many other schemes as well. It follows from (5.4) that λ is non-dimensional. λ is often called the Courant number (after Richard Courant).

Equation (5.3) is called a one-step scheme because v_j^{n+1} depends only on data at time level n. In particular there is no problem in starting from the initial data, i.e., $v_j^0 = g(x_j)$. It is also called an explicit scheme since v_j^{n+1} can be obtained directly from the data at time level n. There is no coupling among the values at time level $n+1$ for different values of j.

If we apply the differencing to the exact solution $u(x,t)$ we get

$$u_t(x_j, t_n) = \frac{u_j^{n+1} - u_j^n}{\Delta t} - \Delta t \; u_{tt} + O(\Delta t^2), \tag{5.5}$$

$$u_x(x_j, t_n) = \frac{u_{j+1}^n - u_{j-1}^n}{\Delta x} - \frac{\Delta x^2}{6}\, u_{xxx} + O(\Delta x^4). \tag{5.6}$$

If we equate the left-hand side of (5.5) to the left-hand side of (5.6) (from the equation (5.2)), and only keep the highest order term in the error, we find that u_j^n (i.e., the exact solution evaluated at the grid points) satisfies the equation

$$u_j^{n+1} = u_j^n + \frac{\lambda}{2}\,(u_{j+1}^n - u_{j-1}^n) + \Delta t T, \tag{5.7}$$

where

$$T = \frac{\Delta t}{2}\, u_{tt} - \frac{\Delta x^2}{6}\, u_{xxx} \tag{5.8}$$

is called the truncation error. It is the amount by which the exact solution fails to satisfy the difference scheme.

In the forward Euler case the truncation error contains a term proportional to Δt and to Δx^2. This should not surprise you since we are using first order differencing in t and second order differencing in x. Schemes where the truncation error is proportional to

$$A\Delta t^p + B\Delta x^q$$

are called (p, q) schemes. Thus forward Euler (5.3) is a (1,2) scheme. We will generalize this concept later. Note some references reverse the order of p and q.

Von Neumann Analysis

What is the analogue of Fourier analysis for a difference scheme? We can certainly assume the x dependence to be of the form $\exp(ikx_j)$, but it is not clear what to do about the t dependence. We saw before that the solution can have constant modulus in t, can decay in t or can blowup in t. In order to account for all possibilities let us assume

$$v_j^n = z^n \exp(ikx_j), \tag{5.9}$$

where z is complex. Note that if $\mid z \mid = 1$ the solution has constant modulus, if $\mid z \mid < 1$ the solution decays in time and if $\mid z \mid > 1$ the solution blows up in time. We make one further simplification of notation. Let $\xi = \exp(ikh)$. Then we have $\mid \xi \mid = 1$ and we can rewrite (5.9) as

$$v_j^n = z^n \xi^j. \tag{5.10}$$

. Now to do the analysis plug (5.10) into the difference scheme (5.3), assume $z \neq 0$ which is the non-trivial case, divide both sides by z^n, solve for z and determine where z is with respect to the unit circle in the complex plane. From simple algebra we get

$$z = 1 + i\lambda \sin(kh). \tag{5.11}$$

Notice that for the exact solution we would have $z = \exp(i\omega\Delta t) = \exp(ik\Delta t)$ so that $\mid z \mid = 1$.

What can we say from (5.11)? Certainly if $kh = 0$, so that the mode is constant in x, then $z = 1$. This is just what you should get from the exact solution. Thus the difference scheme is exact for constants. This is called consistency. It says that the scheme is exact for constants or alternatively it is at least zeroth order accurate.

Next let's see what happens for fixed k as $h \to 0$. We have

$$z = 1 + i\lambda \sin(kh) \simeq 1 + i\lambda kh = 1 + ik\Delta t. \tag{5.12}$$

Thus,

$$z^n \simeq (1 + ik\Delta t)^n \simeq \exp(ikn\Delta t) = \exp(ikt_n).$$

Thus in this limit we recover the exact solution. However, computationally we can see modes where kh is not small. Suppose for example that $kh = \pi/2$. In this case we have

$$z = 1 + i\lambda, \ \mid z \mid = \sqrt{1 + \lambda^2}, \ \mid z \mid > 1,$$

so that z^n explodes as n increases.

This kind of analysis is called von Neumann analysis (after John von Neumann). We will use it heavily. Also note that it is a general theorem that any scheme that is forward in time and centered in space is unstable.

Backward Euler

In the backward Euler scheme we do almost exactly what we did for forward Euler (see (5.3)) except that we compute the spatial derivative at time level $n + 1$. The scheme is

$$v_j^{n+1} = v_j^n + \lambda/2 \ (v_{j+1}^{n+1} - v_{j-1}^{n+1}). \tag{5.13}$$

This is still a one-step scheme (it involves only levels n and $n+1$) and you can easily see that it is a (1,2) scheme. However, there is now a complication. The values at level $n + 1$ for different j, v_j^{n+1} are all coupled together. In order to advance the solution one timestep you must solve a system of linear equations. For example, if we rewrite (5.14) to get all of the terms at level $n + 1$ together we would get the system of equations

$$v_j^{n+1} - \lambda/2 \ (v_{j+1}^{n+1} - v_{j-1}^{n+1}) = v_j^n. \tag{5.14}$$

Now if you were to write this system as a matrix times a vector the j^{th} row would correspond to the equation at grid point x_j. Thus 1 (coefficient of v_j^{n+1}) will be on the diagonals. The $j + 1$ term would correspond to the first diagonal above the main diagonal (first super diagonal) and would have the coefficient of v_{j+1}^{n+1} which is $-\lambda/2$. The first sub-diagonal would have the coefficient of v_{j-1}^{n+1} which is just $\lambda/2$. A matrix for which there are non-zeros only on the main diagonal and the first super diagonal and first sub diagonal is called a tridiagonal matrix. It turns out that it is easy and efficient to solve tridiagonal systems.

However, you may be questioning the size of the system you have to solve. As written here (5.14) is valid for every j. Thus the linear system is infinite and you

can not solve such a system on the computer. The problem is we have considered an infinite interval. If we had considered (5.2) on a finite interval then we would have had a finite system. This would force us to consider boundary conditions which we will do later.

If we assume that we can solve the linear equation we still have to address the question of stability. Is (5.13) stable? You should recognize the importance of this question as you just saw that a perfectly reasonable scheme (forward Euler (5.3) is unstable. We can do the von Neumann analysis by setting $v_j^n = z^n \xi^j$, plugging into (5.14) and getting an equation for z in terms of kh and λ. After a little bit of algebra we get

$$z(1 - i\lambda \sin(kh)) = 1. \tag{5.15}$$

You should be getting pretty good by now in replacing difference quotients by sines.

Clearly if $kh = 0$ we get $z = 1$ so the scheme is consistent. However if we rewrite (5.15) to solve for z we get

$$z = 1/(1 - i\lambda \sin(kh)),$$

and after a little bit of algebra you can see that $\mid z \mid \leq 1$ for all kh so the scheme is unconditionally stable (stable for all λ) and furthermore $\mid z \mid < 1$ unless $kh = 0, \pi$. Thus not only is backward Euler stable but it dissipates most modes (like what you saw before for upwind differencing).

You may think this is completely wrong. Why would you want the numerics to provide a dissipation of energy when there is no such mechanism in the underlying equation? In fact, the dissipation is within the truncation error and in many nonlinear problems backward Euler is very useful because it suppresses instabilities that you can not analyze. Remember in this subject you are always analyzing simple problems and trying to use the numerics for hard problems that you can not analyze.

Since backward Euler is our first stable scheme, it is the first scheme that has a chance of getting the right answer. You can then ask the question, what about convergence? Convergence is the following question. What if h and Δt both $\to 0$ and n and j both $\to \infty$ in such a way that $x_j \to x$ and $t_n \to t$. In this limit we assume that Δt and h approach zero in such a way that the ration λ is fixed. When can you say that $v_j^n \to u(x,t)$. It turns out that there is a general theorem for linear equations which states that if a scheme is stable and consistent it is convergent. We will not go into this, generally the main issues in solving a problem are getting a stable scheme and getting an accurate scheme. If so, for practical purposes you can generally assume that the scheme converges.

This example should show you one more thing. Often seemingly harmless and innocuous changes to the scheme can make a big difference as far as stability is concerned. We will see this with other schemes as well.

$\delta-$formulation

We discuss here a reformulation of backward Euler (and of other implicit schemes) which tends to be better behaved numerically. In the $\delta-$formulation you do not solve directly for v_j^{n+1}. Instead you rewrite (5.13) in terms of

$$\delta_j = v_j^{n+1} - v_j^n,$$

and solve for δ_j. This gives the system of equations

$$\delta_j - \lambda/2\ (\delta_{j+1} - \delta_{j-1}) = \lambda/2\ (v_{j+1}^n - v_{j-1}^n). \tag{5.16}$$

Note that you must also do the update step

$$v_j^{n+1} = v_j^n + \delta_j. \tag{5.17}$$

Why is the $\delta-$formulation worth doing? In many applications you are solving the time dependent equation only to get to the steady state, i.e., a state where the solution is independent of time. So for example if you are solving

$$u_t = u_x$$

you are really interested in the steady state equation

$$u_x = 0.$$

From (5.16) you can see that at steady state (i.e., $\delta_j = 0$) v_j^n solves the steady state equations,

$$v_{j+1}^n - v_{j-1}^n, \tag{5.18}$$

independent of the timestep. Thus the $\delta-$formulation automatically monitors how close you are to steady state and gives you the correct steady state equations.

Another advantage of the $\delta-$formulation is that often the solution is not changing very much. Thus v_j^{n+1} may be very close to v_j^n. If this is so, and you solve directly for v_j^{n+1} you risk that the change δ_j can get swamped by round-off errors. This way you are solving directly for the change.

Lax-Friedrichs

So far we have not come up with a stable explicit scheme. One of the most basic explicit schemes is the Lax-Friedrichs scheme. We do the same thing that we did for forward Euler (see (5.3)), except that we average at level n. The scheme is

$$v_j^{n+1} = \frac{(v_{j+1}^n + v_{j-1}^n)}{2} + \frac{\lambda}{2}\ (v_{j+1}^n - v_{j-1}^n). \tag{5.19}$$

It is easy to see that (5.19) is a (1,2) scheme in the same way as (5.3). You might also think that the averaging in (5.19) should only make a minor change to the

properties of the scheme. However suppose we do a von Neumann analysis. Set $v_j^n = z^n \xi^j = z^n e^{ijkh}$, plug into (5.19) and get an equation for z. The equation is

$$z = \cos(kh) + i\lambda \sin(kh). \tag{5.20}$$

In order to see (5.20) you should by now expect that the central difference term is going to give you something related to $i \sin(kh)$. It is easy to see that the averaging gives you the cosine term.

It follows from (5.20) that $\mid z \mid \leq 1$ provided $\lambda \leq 1$. In fact $\mid z \mid < 1$ if $\lambda < 1$ provided $kh \neq 0, \pi$. Thus Lax-Friedrichs tends to dissipate small disturbances. It tends to be inaccurate, but it tends to be very robust, particularly in dealing with complex, nonlinear problems which may be prone to instabilities.

We make one further point. Suppose that instead of (4.1), you had a sound speed that was different from 1. Specifically suppose we wanted to solve

$$u_t = cu_x, \;\; u(x,0) = g(x), \;\; -\infty < x < \infty. \tag{5.21}$$

Note that c now has units of speed (length/time). You can set up Lax-Friedrichs for (5.21). You now have to use (5.4) for λ and the stability condition is

$$c\Delta t/h \leq 1.$$

Typically, you give h and a value of λ and determine Δt, i.e.,

$$\Delta t = \frac{\alpha h}{c}.$$

The number α is often called the Courant number.

Note that this is another example where a seemingly innocuous change can stabilize an unstable scheme. However, you should realize that generally averaging tends to smooth out disturbances and in many instances is a stabilizing operation. This example also shows the power of the von Neumann analysis in predicting stability for a scheme. Note also that unlike backward Euler (5.13 the Lax-Friedrichs scheme is not unconditionally stable. It is only stable for a range of λ. Furthermore, as a general rule the more implicit you are the more stable you are. While many implicit schemes can be unconditionally stable, explicit schemes can be at most conditionally stable.

You now have the tools to see this, at least heuristically. Suppose first that you have a 1-step explicit scheme. When you do the von Neumann analysis you will get z^{n+1} on the lefthand side of the equation while the righthand side of the equation will depend on z^n and λ. When you cancel out z^n you get a relationship of the form

$$z = H(\lambda, kh),$$

where H is some function. For an explicit scheme λ will be in the numerator. Now unconditional stability means that you can let $\lambda \to \infty$ while z satisfies $\mid z \mid \leq 1$.

Clearly this can not happen. If you have a multi-step scheme, for example a 2 step scheme so that v_j^{n+1} depends on so that v_j^n and v_j^{n-1}, then you cancel z^{n-1} and you get an equation of the form

$$z^2 = H(z, \lambda, kh),$$

where λ is still in the numerator, and at least heuristically you expect that if $\lambda \to \infty$, you will get $z \to \infty$. For implicit schemes, like Backward Euler, you get λ in the denominator and that's why you can get unconditional stability.

Implicit vs. Explicit Schemes

Often when you solve problems you are called upon to make a choice. Explicit schemes are easy to compute, do not require you to work with matrices and are easy to program. However, they have stability bounds and are prone to nonlinear instabilities that go beyond what comes out of the von Neumann analysis. On the other hand implicit schemes tend to be more stable and robust. But they are expensive computationally and require you to work with matrices and use linear algebraic techniques to update the solution.

This tradeoff gets more complicated to evaluate when you consider nonlinear equations. Suppose for example that you have a nonlinear conservation law,

$$u_t = f_x, \quad f = f(u).$$

In this case, it is very straightforward to extend an explicit scheme, such as Lax-Friedrichs. However, it is not easy at all to extend an implicit scheme like backward Euler. In fact, in this case you would get a system of nonlinear equations to solve, which in general would be much harder to solve than the linear system that we considered. We will discuss this later.

Leap Frog

The leap frog scheme is a central difference in both space and time.

$$v_j^{n+1} = v_j^{n-1} + \lambda \, (v_{j+1}^n - v_{j-1}^n). \tag{5.22}$$

Note that (5.22) is a two level scheme. Since we only have one set of initial data, leap frog needs some starting procedure to get v_j^1. Note also that the name is suggestive of the scheme since to get level $n+1$ you leap over level n. Another way to look at this is that the odd time levels are obtained from leaping over the even time levels. Similarly the even time levels leap over the odd time levels. You will see that this is a useful way to look at the scheme.

It is easy to see that (5.22) is a (2,2) scheme. It can be shown that in order to maintain second order accuracy it is sufficient for the starting procedure to be a first order scheme.

We can easily do a von Neumann analysis for leap frog. Set

$$v_j^n = z^n \exp(ikhj). \tag{5.23}$$

If we plug into (5.22) and now divide by z^{n-1} we get a quadratic equation for z. You should easily be able to see that for each additional step of a scheme you get an additional power of z in the von Neumann analysis. Thus if you had a 3 step scheme you would get a cubic polynomial for z.

The equation is

$$z^2 - 2i\lambda \sin(kh) - 1 = 0,$$

and using the quadratic formula we get

$$z = i\lambda \sin(kh) \pm \sqrt{1 - \lambda^2 \sin(kh)^2}. \tag{5.24}$$

You can easily see that $\mid z \mid = 1$ provided $\lambda \leq 1$. You can also easily see that if $\lambda > 1$ one of the roots satisfies $\mid z \mid > 1$. Thus leap frog has the stability bound $\lambda \leq 1$. Remember, if we had a sound speed, i.e., if we solved the equation

$$u_t = cu_x,$$

the stability bound would be

$$\frac{c\Delta t}{h} \leq 1.$$

When leap frog is stable, there is no dissipation. It is a purely non-dissipative scheme. The numerical errors are due completely to dispersion. While you may think this is good, remember that dissipation can serve to damp undesirable modes that can arise from nonlinear interactions. In fact leap frog can be very delicate and is prone to instabilities when used with nonlinear equations.

Leap frog also has the bad property that even for $kh = 0$ (i.e., the constant mode) there are two roots $z = \pm 1$. It is easy to see that the root $z = -1$ is not consistent with a continuous function of time. It represents a decoupling of the odd and even time levels due to the nature of the scheme. Note that since there is no damping, this mode will persist in time. In practice some form of artificial dissipation will have to be introduced. We will discuss this later.

Generally leap frog is run near its stability limit. If you compute the truncation error for leap frog you will find that the truncation error (assuming the basic equation is $u_t = u_x$) is

$$T = \frac{\Delta t^2 u_{ttt} - h^2 u_{xxx}}{6}.$$

Thus when $\lambda \to 1$ (which means $\Delta t = h$) leap frog becomes exact. There is a cancelation of errors as λ approaches 1. This is characteristic of many (2-2) schemes. Generally you never run exactly at the stability limit, but slightly below so as to provide a margin of safety in your computation.

You might think this is great since you can run at bigger timesteps (more efficient calculations) and get more accuracy. However, there are many problems where this can work against you. For example, suppose the equation is

$$u_t = c(x)u_x, \tag{5.25}$$

so that the sound speed is spatially variable. In this case for stability you have to take the smallest λ over the whole domain

$$\lambda \leq \frac{1}{cmax},$$

where $cmax$ is the maximum sound speed over the whole domain. Now suppose that the maximum sound speed occurs at x_1 while for some x_2, $c(x_2) \ll c(x_1)$. Then your computations near x_1 will be very accurate because you are near the stability limit. However, near x_2 the effective λ will be small and the computations will be less accurate. This is a characteristic problem with leap frog and other (2-2) schemes. Large variations in c cause large variations in the effective λ. In practice, after long times the solution will be dominated by errors due to the small λ regions.

(2-4) Leap Frog

There is a very common variant of leap frog that is second order in time and fourth order in space. It is obtained simply by using fourth order central differences in space. You might ask why you don't use fourth order differences in time. The starting procedure with leap frog is hard enough when you have to get one step. Doing fourth order central differences in time is just generally not done. Also generally schemes with many levels in time require a lot of storage.

The (2-4) leap frog is

$$v_j^{n+1} = v_j^{n-1} + \lambda\,(\frac{4}{3}\,(v_{j+1}^n - v_{j-1}^n) - \frac{1}{6}\,(v_{j+2}^n - v_{j-2}^n)). \tag{5.26}$$

By repeating the analysis that led to (5.24) you can show that (5.26) is stable provided

$$\lambda \leq \lambda_0 \simeq 0.728. \tag{5.27}$$

In order to see this, note that if you redo the analysis of (5.24) using fourth order spatial differencing, the term $\sin(kh)$ is replaced by the term

$$F(kh) = \frac{4\sin(kh)}{3} - \frac{\sin(2kh)}{6}.$$

The stability condition is then

$$1 - \lambda^2 F^2(kh) \geq 0.$$

This leads to

$$\lambda \leq \frac{1}{\mid F(kh) \mid},$$

and in order to have stability for all kh we have to take the smallest value of λ, i.e.,

$$\lambda \leq \frac{1}{\max \mid F(kh) \mid},$$

where the maximum is taken over the interval $-\pi \leq kh \leq \pi$. This maximum can be computed numerically leading to (5.27). This analysis also shows that the (2-4) leap frog is non-dissipative; the errors are all dispersive. There is still an odd/even decoupling.

The (2-4) leap frog illustrates several general properties.

1. Generally you do not want the storage required for many levels in time. Thus it is very common to work with schemes that are only second order in time and higher order in space. Note that it does not require additional storage to do higher order spatial differencing.

2. It is typical of (2,4) schemes that the stability limit is reduced from the corresponding (2-2) scheme.

3. To understand why this is natural note that the truncation error behaves like

$$O(\Delta t^2) + O(h^4).$$

   Another way to look at this is that if $\Delta t = \lambda h$ then the error looks like

$$O(h^2) + O(h^4),$$

   and there does not seem to be any benefit in using a (2-4) scheme. The error still decreases as h^2. The only way to get something like fourth order accuracy is to use a small timestep (i.e., λ has to be small) so that the second order temporal error does not dominate the fourth order spatial error. The stability analysis, which gives a reduced stability limit, is trying to tell you that you should reduce the timestep.

4. Generally you run (2-2) schemes near their stability limit. There is no point in taking too small a timestep and in fact as we saw above for leap frog spatial and temporal errors often cancel so (2-2) schemes run best near their stability limit. However, (2-4) schemes are often run below their stability limit to reduce the second order temporal errors.

5. The major advantage of (2-4) schemes occurs in higher dimensions. Suppose for example you can cut your grid in half (increase h by 2) by using fourth order spatial differencing. In 3 dimensions this leads to a decrease by a factor of 8 in the required number of grid points. The fact that you may have to run at a smaller timestep is minor compared to the large reduction in spatial grid points.

To recapitulate - you generally run (2-4) schemes well below the stability limit.

Artificial Dissipation

One of the major problems with the leap frog is that of spatial oscillations. These are mostly associated with the fact that modes with $kh = \pi$ are not damped (also modes with kh near π). These can become very pernicious for nonlinear problems and can lead to instabilities.

In order to deal with these oscillations you can add artificial dissipation to the scheme. Artificial dissipation is based on the idea that even derivatives (with the correct sign) are dissipative. In the crudest form, artificial dissipation would be to just add a term like ϵu_{xx} or $-\epsilon u_{xxxx}$ to the scheme. You might think that this is straightforward, however it can be a problem, particularly for leap frog which can be unstable when applied to dissipative problems.

In order to see what happens consider only the ordinary differential equation

$$dy/dt = -ay. \tag{5.28}$$

As we have seen before (5.28) is a model of what happens when you have a dissipative term. You can get such an equation for example by Fourier transforming the heat equation. Solutions to (5.28) decay in time like $\exp(-at)$.

Suppose we apply leap frog (just central differencing in time) to (5.28). We get

$$v^{n+1} - v^{n-1} = -2a\Delta t v^n. \tag{5.29}$$

Notice that we are maintaining the convention that v is to be used for the numerical solution. Next set $v^n = z^n v_0$ to get

$$z^2 + 2a\Delta t z - 1 = 0. \tag{5.30}$$

If you examine the roots of (5.30) you will see that for small values of $a\Delta t$ the roots behave as

$$z = -a\Delta t \pm 1. \tag{5.31}$$

One of the roots behaves like $1 - a\Delta t$ and is consistent with the exponential decay of the solution. However, the other root (sometimes called the parasitic root) is <u>outside</u> the unit circle. This root will give exponentially growing solutions.

We learn from this that straightforward leap frog (or central differencing in time) is <u>unstable</u> for dissipative problems.

The instability can be removed by <u>lagging</u> the dissipation. In particular suppose we replaced (5.29) by

$$v^{n+1} - v^{n-1} = -2a\Delta t v^{n-1}. \tag{5.32}$$

In this case you can easily see that both roots are inside the unit circle. The same thing happens if you average the dissipation, i.e., if you use

$$v^{n+1} - v^{n-1} = -2a\Delta t (v^{n+1} + v^{n-1})/2. \tag{5.33}$$

The basic point is that leap frog can work with dissipative terms provided they are not taken at time level n.

You should see that this is another example where a seemingly small change to the numerical scheme has a big effect on the performance of the scheme.

We now want to consider methods to artificially add dissipation terms to leap frog. Consider first the (2-2) leap frog. You want to add dissipation with coefficients that vanish as $h \to 0$ so as to maintain convergence. Typically the dissipation for leap frog is taken as $O(h^4)$. Thus for the (2-2) leap frog you typically add a term of the form

$$-\epsilon h^4 v^{n-1}_{xxxx}, \tag{5.34}$$

to the scheme, differenced using central differencing (second order).

Note that ϵ must be obtained from numerical experimentation. Also, the artificial dissipation will lead to a reduction in the maximum stable timestep.

The same thing can be done for the (2-4) leap frog, although in this case the dissipation is taken of order h^6. Specifically you add a term of the form

$$\epsilon h^6 v^{n-1}_{xxxxxx}, \tag{5.35}$$

where you can again use second order central differencing (the fourth order truncation error is still applicable because of the term h^6).

When you have a system, then the dissipation terms should be evaluated for each component of the solution. Furthermore, observe that the dissipative term is large when the solution changes rapidly (high derivatives are large), while the dissipation is small when the solution varies gradually (high derivatives are small).

Issues in implementing leap frog

In dealing with leap frog we have two points to consider.

1. Starting procedure

2. Ameliorating the splitting between successive time levels

The starting procedure can lead to problems. Because leap frog is non-dissipative any errors in the starting procedure can manifest themselves over long times - they will not decay.

In order to start the scheme you need some mechanism to advance the solution to the first time level. Sometimes you have a small time expansion for the solution. That is, while you do not know the exact solution you have an approximation which is valid for small times and you can use this to get the values at Δt.

Another method is to use a one-step scheme. This can be Lax-Friedrichs or even the unstable forward Euler scheme. The one-step schemes are often implemented using a bootstrap procedure. To see how this works let Δt be the time that you want to advance the solution to. Then use a one-step scheme to advance to say $\Delta t/8$.

Then use leap frog to advance to $\Delta t/4$. Then repeat with leap frog to advance to $\Delta t/2$ and finally to Δt. This procedure can reduce temporal errors if you use a first order starting procedure, but also works well even if your starting procedure is second order.

The temporal oscillations are directly due to a decoupling of the odd and even timesteps which is itself related to the two roots that you get from the von Neumann analysis (see (5.24)). In particular, the root $z = -1$ for $kh = 0$ is spurious but can result from numerical errors or from errors in the starting procedure. In order to control these oscillations you should periodically couple the odd and even time levels by using a one-step scheme every M timesteps, where M has to be chosen for each problem.

Another approach is to average the solution periodically. Thus for example suppose you have v_j^{n-1}, v_j^n and v_j^{n-1}. You can then define $v_j^{n+1/2} = (v_j^{n+1} + v_j^n)/2$ and $v_j^{n-1/2} = (v_j^n + v_j^{n-1})/2$ and begin the leap frog for the <u>half</u> timesteps.

Lax-Wendroff

We next consider a family of dissipative schemes. We first discuss a precise definition of dissipation. Consider Lax-Friedrichs (5.19). We saw that generally this scheme tends to dissipate small disturbances as for most values of kh we have $z < 1$ provided $\lambda < 1$ (see (5.20). However, Lax-Friedrichs is not strictly dissipative. You can see this by looking at what happens if $kh = \pi$. In this case there is no dissipation, $z = -1$. In order for a scheme to be truly dissipative we would like a condition whereby $\mid z \mid < 1$ for all $kh \neq 0$. (Generally $z = 1$ for $kh = 0$ by consistency).

The definition commonly used in practice is this. A scheme is dissipative of order $2r$ for a particular value of λ if there exists $\delta(\lambda)$ such that

$$\mid z \mid \leq 1 - \delta \mid kh \mid^{2r}, \;\; \mid kh \mid \leq \pi. \tag{5.36}$$

Note that both Lax-Friedrichs and backwards Euler fail to satisfy (5.36) because of problems at $kh = \pi$.

One scheme that is dissipative (of order 4) is the Lax-Wendroff scheme. Consider the equation

$$u_t = u_x, \tag{5.37}$$

with appropriate initial conditions. In order to derive the scheme, recall that u_{xx} is a dissipative term, so we would like to involve this term in the scheme as much as possible. The Lax-Wendroff scheme accomplishes this by employing a Taylor series in t. Consider the expansion for the exact solution

$$u_j^{n+1} = u_j^n + \Delta t\, u_{t,j}^n + \frac{\Delta t^2\, u_{tt,j}^n}{2} + O(\Delta t^3). \tag{5.38}$$

Now use (5.37) to replace u_t by u_x and u_{tt} by u_{xx}. Then use central differences to

approximate u_x and u_{xx}. We get the scheme

$$v_j^{n+1} = v_j^n + \frac{\lambda}{2}(v_{j+1}^n - v_{j-1}^n) + \frac{\lambda^2}{2}(v_{j+1}^n + v_{j-1}^n - 2v_j^n). \tag{5.39}$$

Recall that we are using v to denote numerical solutions. Note that (5.39) differs from forward Euler by the last term. We expect this term to be stabilizing because it approximates u_{xx}. It is easy to see that Lax-Wendroff is a (2-2) scheme. Note, that Lax-Wendroff schemes are complicated in that the temporal and spatial differences are coupled together. You can't directly separate out a part due to temporal differences from a part due to spatial differences. This is different from simpler schemes like leap frog or Lax-Friedrichs.

We next do a von Neumann analysis for (5.39). Set $v_j^n = z^n \exp(ikhj)$ and plug into the scheme. After some algebra you can see that you get

$$z = 1 + i\lambda \sin(kh) + \lambda^2(\cos(kh) - 1). \tag{5.40}$$

It is easy to see that $\mid z \mid \leq 1$ provided $\lambda \leq 1$. Note that $z = 1 - 2\lambda^2 < 1$ for $kh = \pi$. Furthermore, after some more involved algebra, you can see that

$$\mid z \mid \leq 1 - \delta \mid kh \mid^4,$$

provided $\lambda < 1$, so that Lax-Wendroff is <u>dissipative</u> of order 4. Note that the dissipation vanishes as $\lambda \to 0$. This can sometimes cause instabilities when there are nonlinear problems and regions of space where the effective λ is close to 0.

You now have two ways to get some dissipation into your scheme. With leap frog we used <u>artificial</u> dissipation. This required us to determine ("tune") a parameter ϵ. With Lax-Wendroff, dissipation is built into the scheme. We have no mechanism to control the dissipation. Both approaches have advantages and disadvantages. For example, the ability to control the amount of dissipation means that you can fine tune your computation. You might, for example, make ϵ a function of x so that there is enhanced dissipation in regions that are causing you trouble (giving rise to instabilities). On the other hand, artificial dissipation will generally require several runs to get the amount of dissipation right. It can also lead to a reduction in the timestep. These "tuning" runs are avoided by using a dissipative scheme such as Lax-Wendroff. Of course you can always add more dissipation to any scheme, however in using Lax-Wendroff or any other dissipative scheme you generally have no mechanism to <u>reduce</u> the amount of dissipation built into the scheme.

MacCormack's Scheme

We will see below that the Lax-Wendroff scheme can be difficult to implement for nonlinear and variable coefficient problems. The problem is in using the equation to compute the analogue of u_{tt}. A scheme that avoids this problem is the MacCormack scheme. It is a 1-step scheme, although divided into two half-steps called the predictor

and corrector stems. There are also two variants of the MacCormack scheme. We consider one variant, sometimes called the FB variant because there is a forward predictor and a backwards corrector. The scheme is

$$\hat{v}_j = v_j^n + \lambda(v_{j+1}^n - v_j^n), \tag{5.41}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \lambda(\hat{v}_j - \hat{v}_{j-1})). \tag{5.42}$$

Thus you make a prediction ($\hat{v}_j$) using a forward predictor (5.41) and then correct this value in the corrector (5.42). Using a little bit of algebra you can see that for (5.37) the MacCormack scheme (5.41-5.42) is equivalent to the Lax-Wendroff scheme. Note in particular that even though you are using combinations of one-sided differences the result is equivalent to the Lax-Wendroff scheme in which points $j+1$ and $j-1$ are treated symmetrically. It is important to realize that MacCormack is not an upwind scheme and you do not change the direction of the differencing if the direction of the characteristics changes.

It is easy to see that there is a variant of MacCormack which uses a backwards predictor and a forward corrector. This is also identical to Lax-Wendroff for (5.37). In particular both variants of MacCormack are dissipative.

The MacCormack scheme is very widely used in practice. We first indicate how to extend it to more general equations. First consider the conservation law

$$u_t = f_x \tag{5.43}$$

where $f(u)$ is some given function. We can derive a Lax-Wendroff type scheme for (5.43) although it is complicated to get an approximation to u_{tt} (see (5.38). In particular we have

$$u_{tt} = f_{xt} = f_{tx} = (f'u_t)_x = (f'f_x)_x.$$

The problem is you have to compute f'. If you had a vector function then f' would be replaced by the Jacobian matrix. Thus in this case Lax-Wendroff could become extremely complicated. In contrast, the FB variant of MacCormack scheme is simply

$$\hat{v}_j = v_j^n + \lambda(f(v_{j+1}^n) - f(v_j^n)), \tag{5.44}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \lambda(f(\hat{v}_j) - f(\hat{v}_{j-1}))). \tag{5.45}$$

For the case that $f(u)$ is a nonlinear function of u it is not the case that the FB and BF variants are identical. Furthermore they are not equivalent to Lax-Wendroff. However, they are both robust and dissipative schemes. For nonlinear problems it is preferable to alternate the FB and BF variants. A reasonable stability bound to employ is

$$C\Delta t/\Delta x \leq 1,$$

where

$$C = \max \mid f'(v_j^n) \mid$$

and the maximum is taken over all grid points j. Thus for nonlinear equations you may have to adjust the timestep as time evolves. In order to see where this comes from recall that if the equation were

$$u_t = cu_x,$$

where c is a constant sound speed then the stability limit would be

$$c\Delta t/h \leq 1.$$

In the case of a conservation law we can rewrite (5.43) as

$$u_t = f'(u)u_x,$$

and thus $f'(u)$ plays the role of a nonlinear sound speed. For stability you have to take the smallest stable timestep which means taking the largest local velocity.

You can verify that if $u(x,t)$ is a solution to (5.43 then the solution would satisfy either of the MacCormack variants to second order accuracy. Furthermore, if you have variable coefficients it is sufficient to evaluate them at the point x_j.

In implementing MacCormack, it is generally preferred to write subroutines directly for the FB and BF versions and then just alternate calling them in the main program. You can also alternate in one subroutine but this makes the programming more complicated and more prone to bugs.

Note that we have given the MacCormack scheme for equations which do not depend explicitly on t. If the equation does depend on time then to get the second order accuracy in time the righthand side for the predictor should be evaluated at time level n and for the corrector the righthand side should be evaluated at level $n+1$. You can easily verify this by neglecting the x dependence and looking only at an ordinary differential equation in time.

(2-4) MacCormack

It is possible to extend the MacCormack formulation to develop (2-4) dissipative schemes. However, we will have to modify our definition of order of accuracy.

A scheme is said to be of order (p,q) if the truncation error E satisfies

$$E = \Delta t F(\Delta t, \Delta x), \tag{5.46}$$

where

$$F(\Delta x^{q/p}, \Delta x) = O(\Delta x^q)$$

Note that we will use here Δx instead of h to make the concepts more transparent. To see what this means set $p = 2$ and $q = 4$. This says that if $\Delta t = O(\Delta x^2)$ then the

error is $O(\Delta x^4)$. Thus you will get fourth order accuracy in x, but you can no longer have Δt scaling linearly with Δx. Clearly if the original truncation error satisfies

$$F(\Delta t, \Delta x) = O(\Delta t^p + \Delta x^q),$$

for example the (2-4) leap frog, then the original definition is equivalent to the new definition. The new definition is necessary to deal with schemes like MacCormack where the time differencing is intertwined with the spatial differencing.

With this definition there is a family of (2-4) extensions of the MacCormack scheme. The FB variant for the most commonly used one is

$$\hat{v}_j = v_j^n + \frac{\lambda}{6}(-v_{j+2}^n + 8v_{j+1}^n - 7v_j^n), \tag{5.47}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \frac{\lambda}{6}(7\hat{v}_j - 8\hat{v}_{j-1} + \hat{v}_{j-2})). \tag{5.48}$$

Note that both predictor and corrector are only first order accurate. From consistency the sum of the coefficients inside each of the difference terms has to be 0.

The BF variant is

$$\hat{v}_j = v_j^n + \frac{\lambda}{6}(7v_j^n - 8v_{j-1}^n + v_{j-2}^n), \tag{5.49}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \frac{\lambda}{6}(-7\hat{v}_j + 8\hat{v}_{j+1} - \hat{v}_{j+2})). \tag{5.50}$$

For constant coefficient problems you can show that each variant has a truncation error of the form

$$E = \Delta t(O(\Delta x^4) + O(\Delta t \Delta x^2) + O(\Delta t^2)).$$

(This requires some messy algebra but is straightforward). Thus each scheme is a (2-4) scheme. For nonlinear problems you can only get fourth order accuracy by alternating the FB and BF variants. In practice, even for constant coefficient problems these variants should be alternated. If you have a conservation law like (5.43) then the FB variant is

$$\hat{v}_j = v_j^n + \frac{\lambda}{6}(-f_{j+2}^n + 8f_{j+1}^n - 7f_j^n), \tag{5.51}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \frac{\lambda}{6}(7f(\hat{v}_j) - 8f(\hat{v}_{j-1}) + f(\hat{v}_{j-2}))). \tag{5.52}$$

The BF variant is similar. If you have variable coefficients you can evaluate them at x_j.

The stability bound can be worked out but it is very messy. The result is that the (2-4) MacCormack is stable provided

$$\lambda = \Delta t/\Delta x < 0.67.$$

In practice the scheme must be run considerably below the stability limit to see the benefits of the fourth order accuracy. However the scheme is very much more accurate than the (2-2) MacCormack and much more robust than the (2-4) leap frog. Thus it is a practical fourth order scheme.

This methodology can be extended to (2-6) schemes. One such scheme is (we only give the FB variant)

$$\hat{v}_j = v_j^n + \frac{\lambda}{30}(-37v_j^n + 45v_{j+1}^n - 9v_{j+2}^n + v_{j+3}^n), \tag{5.53}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \frac{\lambda}{30}(37\hat{v}_j - 45\hat{v}_{j-1} + 9\hat{v}_{j-2} - \hat{v}_{j-3}^n)). \tag{5.54}$$

To verify that (5.53-5.54) is a (2-6) scheme requires some messy but straightforward algebra. The stability bound has not been worked out precisely but is approximately $\lambda < 0.3$. In practice the timestep has to be reduced considerably in order to get the benefits of the sixth order accuracy.

Runge-Kutta

Schemes based on Runge-Kutta time differencing have become widely used over the past few years. Runge-Kutta schemes are somewhat analogous to MacCormack type schemes in that there are several stages that have to be computed. There are many different Runge-Kutta schemes. We consider the most popular Runge-Kutta scheme, namely four stage, fourth order Runge-Kutta. We will also consider a low storage version. We first describe the scheme for an ordinary differential equation.

Consider the equation

$$du/dt = f(u,t). \tag{5.55}$$

Note that there is now no spatial dependence, (5.55) is just an ordinary differential equation. Suppose you have a solution at time level n. Following our previous notation we will call the numerical solution v^n. (There is now no j since there is no x dependence). In fourth order Runge-Kutta you do the following stages

$$K_1 = \Delta t f(v^n, t_n), \quad \hat{v}_1 = v^n + \frac{K_1}{2}, \tag{5.56}$$

$$K_2 = \Delta t f(\hat{v}_1, t_n + \Delta t/2), \quad \hat{v}_2 = v^n + \frac{K_2}{2}, \tag{5.57}$$

$$K_3 = \Delta t f(\hat{v}_2, t_n + \Delta t/2), \quad \hat{v}_3 = v^n + K_3, \tag{5.58}$$

$$K_4 = \Delta t f(\hat{v}_3, t_n + \Delta t), \quad v^{n+1} = v^n + (K_1 + 2K_2 + 2K_3 + K_4)/6. \tag{5.59}$$

This scheme requires 4 stages. You can show that it is fourth order accurate in t for any f. The analysis is very messy but if f is only a function of t you can see easily that this reduces to Simpson's rule for integration.

Before discussing partial differential equations, we want to analyze this scheme and prepare ourselves for a von Neumann analysis. We still restrict to ordinary differential equations. We consider the specific equation

$$du/dt = au, \quad u(0) = u_0. \tag{5.60}$$

The exact solution is

$$u(t) = \exp(at)u_0. \tag{5.61}$$

Suppose we do Runge-Kutta and look for solutions of the form

$$v^n = z^n u_0, \tag{5.62}$$

(Note the similarity to von Neumann analysis). In order to find z you just have to do one step of Runge-Kutta with u_0 as initial condition. If you do this you will see that

$$z = P_4(a\Delta t), \tag{5.63}$$

where P_4 is a fourth degree polynomial. In order to see why this is true just look at the form of (5.56-5.59). You can easily see that each stage gives you a linear factor of $a\Delta t$ and since you have four stages you will get a fourth order polynomial.

Now you might think that it would be a mess to evaluate the coefficients of this polynomial. However, it is not necessary to do the algebra. If you accept what I said before, that (5.56-5.59) is a fourth order scheme and remember that the exact solution at time Δt is $\exp(a\Delta t)u_0$ (from (5.61)), then you can see that P_4 must be a fourth order polynomial which is a fourth order approximation to $\exp(a\Delta t)$. From the Taylor series for $\exp(at)$ we have

$$\exp(a\Delta t) = 1 + (a\Delta t) + (a\Delta t)^2/2! + (a\Delta t)^3/3! + (a\Delta t)^4/4! + O((a\Delta t)^5). \tag{5.64}$$

It follows from (5.64) that the first five terms in the expansion,

$$P_4(a\Delta t) = 1 + (a\Delta t) + (a\Delta t)^2/2! + (a\Delta t)^3/3! + (a\Delta t)^4/4! \tag{5.65}$$

is such a fourth order polynomial. Furthermore it is easy to see that it is the only such polynomial. Any other polynomial must differ from $P_4(a\Delta t)$ by terms of at most $O((a\Delta t)^4)$ while both must agree with $\exp(a\Delta t)$ up to order $O((a\Delta t)^5)$ and both conditions can not be satisfied.

Thus given the fact that Runge-Kutta is fourth order we have been able to do the analogue of von Neumann analysis. Before doing a partial differential equation we will use what we have just derived for a special case. Suppose a is purely imaginary. Let's write $a = i\tilde{a}$. In this case solutions to (5.60) are bounded (you should be thinking of hyperbolic equations where the righthand side is always imaginary). We can then ask the question, for what values of Δt will Runge-Kutta be stable. From what we have already done and your experience with von Neumann analysis, you should be

able to answer this question. Simply look at $z = P_4(i\tilde{a}\Delta t)$ and find conditions for $\mid z \mid \leq 1$. We can phrase this in a simple way. Look at the polynomial $P_4(w)$ for w on the imaginary axis. Suppose that you find the largest value of $\mid w \mid$ such that $\mid P_4(w) \mid \leq 1$. Specifically suppose you find the largest R such that $\mid P_4(w) \mid \leq 1$ for $\mid w \mid \leq R$. Then Runge-Kutta is stable provided

$$\mid i\tilde{a}\Delta t \mid \leq R, \tag{5.66}$$

which we can write as

$$\Delta t \leq R/ \mid a \mid . \tag{5.67}$$

R can be determined from simple computer experiments, $R \simeq 2.8$. Thus, the scheme (5.56-5.59) is stable provided

$$\Delta t \leq 2.8/ \mid a \mid .$$

We next discuss how to apply Runge-Kutta to a partial differential equation. We will assume that you are doing fourth order differencing in space. Suppose we let D_4 denote the fourth order central difference operator (3.15). Furthermore suppose we define $\vec{v}$ as the vector containing the solution v_j at each grid point. Then we can convert the equation $u_t = u_x$ into the system of ordinary differential equations

$$dv_j/dt = D_4\vec{v} \mid_j, \tag{5.68}$$

where j varies over the grid. (Note that v_j is only a function of t, so we use the notation for ordinary derivative.) Of course in practice we would need either boundary conditions or periodicity to close the system. Finally we can apply the scheme (5.56-5.59) to the system (5.68). Note that you can do this for more general equations as well. In particular you can deal with time dependent coefficients as well as spatially dependent coefficients.

Now all of the machinery that we developed above for ordinary differential equations can be applied to help you do the von Neumann analysis. We will do the von Neumann analysis in a slightly different manner from what we have done before just to simplify the presentation. We first do a Fourier analysis in x. If we set

$$v_j = a(t)\exp(ikhj)\vec{v_0},$$

and plug into (5.68) we get an ordinary differential equation for $a(t)$,

$$da/dt = i(\frac{4}{3}\sin(kh)/h - \frac{1}{3}\sin(2kh)/(2h)) = iF_4(kh)/h \tag{5.69}$$

where we define

$$F_4(kh) = \frac{4}{3}\sin(kh) - \frac{1}{6}\sin(2kh),$$

(refer back to (3.16) and the argument that led to (5.27)). It then follows from (5.67) that fourth order Runge-Kutta is stable for any particular kh provided

$$\mid F_4(kh)\Delta t/h \mid \leq R,$$

or

$$\Delta t \leq h(R/F_4(kh)). \tag{5.70}$$

You now have to do this for all kh. In order to get stability for all Fourier modes you have to take the smallest timestep for all kh which means you have to compute the maximum of $\mid F_4(kh) \mid$ for $-\pi < kh \leq \pi$. You can compute this maximum numerically to get

$$\Delta t \leq h(0.728)(2.8).$$

You should refer to the analysis for the (2-4) leap frog and observe that you get the same function to maximize.

This version of Runge-Kutta is fully fourth order, even for problems where the coefficients depend on time. It is storage intensive however, as each stage must be stored. This can be a problem in 3 dimensions. A low storage version is also used for problems which are independent of time. We will give this version for the ordinary differential equation

$$du/dt = f(u),$$

$$K_1 = \Delta t f(v^n), \quad \hat{v}_1 = v^n + K_1/4, \tag{5.71}$$

$$K_2 = \Delta t f(\hat{v}_1^n), \quad \hat{v}_2 = v^n + K_2/3, \tag{5.72}$$

$$K_3 = \Delta t f(\hat{v}_2^n), \quad \hat{v}_3 = v^n + K_3/2, \tag{5.73}$$

$$K_4 = \Delta t f(\hat{v}_3^n), \quad v^{n+1} = v^n + K_4. \tag{5.74}$$

The advantage of this method is that the individual computations for each stage need not be saved until the end. This method is only fourth order provided the right hand side is independent of t. You can see that this method has exactly the same polynomial as for (5.56-5.59) and thus the same stability bound. It is called low-storage Runge-Kutta.

Generally Runge-Kutta can allow for higher accuracy in time at the cost of additional storage. The scheme is not dissipative for $kh = \pi$ as you can readily see. This leads to spatial oscillations which are generally damped using artificial dissipation in the same way as is done for the leap frog (see (5.34) and (5.35)), although now the actual time level at which the dissipation is introduced is not crucial.

6 SYSTEMS

Before proceeding to a study of implicit schemes we consider the extension of what we have done to systems of equations. Consider a general system of equations

$$\vec{u}_t = A\vec{u}_x, \quad -\infty < x < \infty, \quad \vec{u}(0,x) = \vec{u}_0(x), \tag{6.1}$$

where $\vec{u}, \vec{u}_0$ are $m-$vectors and A is an $m \times m$ matrix. In this section it will be important to distinguish between column and row vectors. In (6.1) $\vec{u}$ is a column vector. All of the schemes that we have so far considered can be applied to (6.1). The only difference is programming. This is true even if A depends on x, t and $\vec{u}$. Note further that all schemes can be applied to systems of conservation laws as well.

What we want to consider here is the effect on stability of having a system instead of a scalar equation. We therefore assume that A is a constant matrix. Remember the definition of hyperbolicity. A has m real and distinct eigenvalues. Let's call these eigenvalues $\mu_1, \mu_2 \dots \mu_m$. Since the eigenvalues are all distinct, the matrix A can not have any Jordan blocks. A must be similar to a diagonal matrix which has the μ's along the diagonal. These eigenvalues have units of length/time. Thus they are speeds. They are sometimes called the characteristic speeds of the system.

We next focus on stability for systems. We illustrate the concepts with leap frog. You should not think that leap frog is special. The analysis we will give is true for all of the schemes that we have studied. We use leap frog only for illustrative purposes. Leap frog for the system (6.1) results in the scheme

$$\vec{v}_j^{n+1} = \vec{v}_j^{n-1} + \lambda A \ (\vec{v}_{j+1}^n - \vec{v}_{j-1}^n), \tag{6.2}$$

where we use the notation $\vec{v}$ to denote the numerical solution. Note that only λA appears in the scheme. We now set $\vec{v}_j^n = z^n \exp(ikhj)\vec{v}_0$ and plug into (6.2). Comparing with (5.23) we see that the difference between the scalar and vector case is the vector $\vec{v}_0$ which has to be determined. We get from (6.2) the vector equation

$$(z^2 I - 2i \sin(kh)\lambda A - I)\vec{v}_0 = 0, \tag{6.3}$$

where I is the identity matrix. Since we do not want $\vec{v}_0$ to be 0, we now get the equation for z by requiring the determinant of the matrix in (6.3) to vanish.

At first glance this looks much more complicated than the scalar case. However, we can reduce it to the scalar case by hyperbolicity. There is a matrix P such that

$$PAP^{-1} = \Lambda \tag{6.4}$$

where Λ is the matrix with the characteristic speeds μ_i on the diagonal. Since the determinant is invariant under similarity transforms, we can transform the matrix problem in (6.3) to the equation

$$\det(z^2 I - 2i \sin(kh)\lambda\Lambda - I) = 0. \tag{6.5}$$

Since this is a diagonal matrix it follows that at least one of the diagonal entries must vanish. It follows that for any λ there are $2m$ values of z, which we denote by $z_i^{\pm}, (i = 1, \dots, m)$ where for each i, $z_i^{\pm}$ are the roots of the quadratic

$$z^2 - 2i \sin(kh)\lambda\mu_i - 1 = 0.$$

By what we have already done for the scalar case we know that $\mid z_i^{\pm} \mid \leq 1$ if and only if

$$\lambda \mid \mu_i \mid \leq 1,$$

(in fact in this case $\mid z_i^{\pm} \mid = 1$.) In order to have stability you have to take the smallest λ or

$$\lambda \leq 1/ \mid \mu_{max} \mid, \tag{6.6}$$

where μ_{max} is the characteristic speed of largest magnitude. This gives the stability bound

$$\Delta t \leq \Delta x/ \mid \mu_{max} \mid . \tag{6.7}$$

This says that the stability bound for the system is the same as for the equation

$$u_t = \mu u_x,$$

with μ replaced by the eigenvalue of A which has largest magnitude. It is easy to see that this argument is general, i.e., valid for any difference scheme.

We restate this another way. For any scheme, first work out the stability bound for the equation

$$u_t = u_x.$$

Call this stability bound w_0. Then for a system of equations, the resulting scheme is stable provided

$$\Delta t \leq w_0 \Delta x/ \mid \mu_{max} \mid . \tag{6.8}$$

Thus stability is determined by λ_0, which is independent of the system or of the coefficients, together with the maximum characteristic speed. Of course if the coefficients are variable or nonlinear, you must use the largest characteristic speed over the whole domain at time level n.

We have concentrated on stability, however we also want to introduce some terminology for the dependent variables. Consider the matrix P defined in (6.4). If we take the original PDE system (6.1) and make the change of dependent variable

$$\vec{w} = P\vec{u}, \tag{6.9}$$

then it is easy to see $\vec{w}$ satisfies the diagonal system of equations

$$\vec{w}_t = \Lambda \vec{w}_x. \tag{6.10}$$

Thus, the transformation (6.9) decouples the dependent variables and reduces the coupled system (6.1) to a system of independent scalar equations. Said another way the coupled system (6.1) is equivalent to a set of scalar equations under the transformation (6.9). (In practice the system will still be coupled, but by boundary conditions which we will discuss later.)

Now let $\vec{p}_i^T$ be the i^{th} row of the matrix P. We use the superscript T to indicate that $\vec{p}_i^T$ is a row vector rather than a column vector. From the similarity transformation (6.4) and the change of variables in (6.9)it is easy to interpret what $\vec{p}_i^T$ represents. In terms of the decoupled variables w_i we have from the rules of matrix times vector multiplication

$$w_i = \vec{p}_i^T \cdot \vec{u},$$

i.e., w_i is simply the standard dot (scalar) product of the i^{th} row of the matrix P with the column vector $\vec{u}$. Furthermore, if you rewrite (6.4) as

$$PA = \Lambda P, \tag{6.11}$$

then it is easy to see that $\vec{p}_i^T$ satisfies the equation

$$\vec{p}_i^T A = \mu_i \vec{p}_i^T. \tag{6.12}$$

From the equation (6.12) you can see why they call $\vec{p}_i^T$ a left eigenvector of the matrix A. This may look strange to you, but it is the standard eigenvalue equation written in terms of row vectors and multiplication on the left instead of column vectors and multiplication on the right.

Now from (6.10) the scalar variable w_i, the i^{th} component of $\vec{w}$ satisfies the wave equation

$$\frac{\partial w_i}{\partial t} = \mu_i \frac{\partial w_i}{\partial x}, \tag{6.13}$$

where we use the symbol ∂ to denote partial differentiation simply to avoid using two subscripts. Thus, the variables that decouple the original system (6.1) are just the result of taking the dot product between the left eigenvectors of A and the original column vector $\vec{u}$. These variables are called the characteristic variables and are very important for the analysis and interpretation of hyperbolic systems. They are just the variables that can be interpreted as waves traveling with speed μ_i. Again, while they can be decoupled for the full space problem (i.e., the problem specified on $-\infty < x < \infty$, in practice they are coupled by the boundary conditions.

It is important to see what the name characteristic means. It is clear from (6.13) that

$$w_i(t, x) = f_i(\mu_i t + x),$$

where $f_i(x)$ is the initial condition for w_i. Thus, the value for the characteristic variable w_i is constant along the characteristic curves in the $t - x$ plane given by

$$\mu_i t + x = \text{constant}.$$

The hyperbolic equation (6.13) serves to propagate the initial condition for the characteristic variable along the characteristic curves as time increases. In practice, of course, things are not so straightforward since variable coefficients may lead to curved characteristics and we will see later that some characteristics can cross for nonlinear

problems. Furthermore, if you have a forcing term then the characteristic variable will no longer be constant along characteristics. However, this is the natural extension of the notion of a characteristic from a scalar equation to a system.

Finally, you should realize that if the matrix A is symmetric, the left eigenvector is just the transpose of the ordinary eigenvector. In this case you can get the characteristic variables by just computing the eigenvectors of the matrix A.

Disparate Speeds

We describe here what can happen if a system has characteristic speeds that are greatly different. This phenomena occurs in may application areas, for example elasticity and magneto-hydrodynamics. Suppose $m = 2$ so that A is a 2×2 matrix with eigenvalues μ_1 and μ_2. Further suppose $\mu_1 \gg \mu_2$ so that μ_1 corresponds to fast waves while μ_2 corresponds to slow waves. Such a situation occurs in elasticity where there are fast compressional waves and slow shear waves. An analogous effect can occur in acoustics (electromagnetism) if there are regions where the sound speed (speed of light) is large and regions where it is small (see (5.25) and the discussion in the section on the leap frog). Here, however, we will assume the constant coefficient case with two different speeds.

Now the equation

$$\vec{u}_t = A\vec{u}_x, \tag{6.14}$$

can be decoupled by working with the new dependent variable $\vec{w} = P\vec{u}$ where P is the matrix that diagonalizes A. If we do this we would get two decoupled scalar equations

$$w^1_t = \mu_1 w^1_x, \quad w^2_t = \mu_2 w^2_x, \tag{6.15}$$

where w^1 and w^2 correspond to the fast and slow waves respectively. Keep in mind that while these waves can be decoupled mathematically, in practice they will be coupled by boundaries or two-dimensional effects and so you solve the coupled system.

Before we always considered what happened when there was a given wave number in x (which we always call k). However, in may problems it makes more sense to consider what happens when there is a given frequency ω. This happens for example, when the system is excited by an external stimulus, say a pulse-like source with peak frequency ω. You can solve each of the equations in (6.15) explicitly to get

$$w^1 = \exp(i\omega(t + x/\mu_1)), \quad w^2 = \exp(i\omega(t + x/\mu_2)). \tag{6.16}$$

You can easily derive this if each wave is to oscillate with frequency ω. Now from (6.16) you can see the important fact. For a fixed ω slower speeds correspond to shorter wavelengths. The wavelengths for the two functions in (6.16) are

$$\lambda_1 = \frac{2\pi\mu_1}{\omega}, \quad \lambda_2 = \frac{2\pi\mu_2}{\omega},$$

respectively. Thus, the waves corresponding to μ_1 are over-resolved in space but since the largest speed (μ_1) determines the timestep, these waves are run with an effective λ near the stability limit. A (2-2) scheme should work fine on them. On the other hand, spatial resolution has to be based on the slow waves (μ_2) which have the smaller wavelengths. However, the effective λ for these waves is small, again since the timestep is controlled by the maximum speed. You can easily see that the effective λ for the (hard to resolve) slow waves is not large, but is rather reduced by the factor μ_2/μ_1. Thus, not only do you have to spend most of your resolution to resolve the slow waves, but the effective λ for these waves is small, so that (2-2) schemes have large truncation errors. This situation is ideal for a (2-4) scheme. In this case you can (and often should) run at a large time step. While (2-4) schemes are less accurate when run near their stability limit, the timestep is set by the fast waves which are over-resolved to begin with. So you don't care if the overall accuracy is only second order. On the other hand, running your (2-4) scheme near the stability limit still results in a small effective λ for the slow waves, and thus you will see the benefits of a (2-4) scheme. At some point, the fast wave timestep restriction will be so severe that you may want to go to an implicit scheme. Note again, that even for problems with one wave speed this problem can occur, for example for interface problems where there are regions of disparate sound or light speeds.

7 IMPLICIT SCHEMES

We have already encountered the backward Euler scheme. You should expect that implicit schemes will have enhanced stability, in fact, many implicit schemes are unconditionally stable. Note, that for many hyperbolic problems this is not so useful. If a solution behaves like $f(\mu t + x)$ where μ is one of the characteristic speeds, then clearly for accuracy you want $\mu\Delta t$ to be comparable to h. However, we saw when we considered problems with disparate speeds, there are problems where there can be be a lot of advantages in exceeding an explicit stability bound. Furthermore, we will see that implicit schemes are very useful for parabolic type problems. Thus, we will consider families of implicit schemes.

A (2-2) implicit scheme can be obtained by evaluating the time derivatives at level $n + 1/2$. This scheme is called the Crank-Nicolson scheme. For the equation

$$u_t = u_x,$$

the scheme is

$$\frac{(v_j^{n+1} - v_j^n)}{\Delta t} = \frac{1}{2}\frac{(v_{j+1}^{n+1} - v_{j-1}^{n+1})}{2h} + \frac{1}{2}\frac{(v_{j+1}^n - v_{j-1}^n)}{2h}. \tag{7.1}$$

Note that the spatial derivatives (righthand side of (7.1)) are centered at level $n+1/2$

by averaging of levels n and $n+1$. It is easy to verify that this, combined with the second order spatial differencing, makes Crank-Nicolson a (2-2) scheme.

If you do a von Neumann analysis you will get the equation

$$z(1 - i\frac{\lambda}{2}\sin(kh)) = 1 + i\frac{\lambda}{2}\sin(kh). \tag{7.2}$$

You can easily see from (7.2) that $\mid z \mid = 1$ for all λ. Thus Crank-Nicolson is unconditionally stable and completely non-dissipative. As with backward Euler, assembling the scheme into a matrix leads to a tridiagonal system of equations which will be closed with boundary conditions. Again as with Backward Euler, the scheme can (and generally should) be reformulated using the $\delta-$formulation (see (5.16)).

In comparing the Crank-Nicolson scheme with the backward Euler scheme note that

1. Backward Euler is dissipative (except for kh near π) but only first order in time.

2. Crank-Nicolson is second order in time, but since it is non-dissipative it is prone to oscillations and nonlinear instabilities.

A family of schemes can be derived which to some extent combines the good and bad features of backward Euler and Crank-Nicolson. This family is defined by the formula

$$\frac{(v_j^{n+1} - v_j^n)}{\Delta t} = \alpha\frac{(v_{j+1}^{n+1} - v_{j-1}^{n+1})}{2h} + (1-\alpha)\frac{(v_{j+1}^n - v_{j-1}^n)}{2h}. \tag{7.3}$$

Note that if $\alpha = 1/2$ we get Crank-Nicolson while with $\alpha = 1$ we get backward Euler. The family (7.3) is second order only for $\alpha = 1/2$. However, there is some dissipation for any $\alpha > 1/2$. Thus by running your program you can get some dissipation by having α close to but greater than $1/2$. In fact, you can formally get second order accuracy by setting

$$\alpha = 1/2 + \alpha_1\Delta t$$

for $\alpha_1 > 0$. You should understand that this is mostly a formal statement. In practice you don't let $\Delta t \to 0$, but in fact you run with only one or at most only a small number of different values of Δt. The family (7.3) should also be implemented in the $\delta-$formulation. Generally you do not build a particular value of α into your program. Rather you write the program with α as a variable and then set it as data when you run the program.

Compact Implicit Schemes

While you can construct (2-4) implicit schemes analogous to Crank-Nicolson there is a problem with these schemes. Standard fourth order differencing involves two sets of neighbors ($j \pm 1$ and $j \pm 2$). When you assemble these terms in the matrix this means you no longer have a tridiagonal matrix, but rather a pentadiagonal matrix. These matrices are much more expensive to invert than tridiagonal matrices.

You might think there is nothing you can do about this. After all, if we restrict to standard finite differences then you need at least 2 neighbors on each side to get fourth order differences. It turns out however that you can get fourth order accuracy provided you take combinations of both the function and its derivative. This will give you a fourth order scheme with only a 3 point stencil, however the scheme will be implicit.

In order to see how this works consider the standard second order central difference formula keeping the leading order term of the error (see (3.3)

$$u_x = (u(x+h) - u(x-h))/(2h) - \frac{h^2}{6}u_{xxx} + O(h^4).$$

In order to simplify the notation let us introduce the notation D_2 for the second order central difference so we can write

$$u_x = D_2 u - \frac{h^2}{6}u_{xxx} + O(h^4). \tag{7.4}$$

Now if we were doing just central differences we would drop the u_{xxx} term and say we have a second order approximation. However, if we can somehow account for this term we would have a fourth order approximation. Furthermore, we only need a second order approximation to the u_{xxx} term because it is multiplied by h^2.

Let $\tilde{D}_2$ be the second order approximation to u_{xx} (see 3.24)

$$u_{xx}(x_j) = \frac{(u_{j+1} + u_{j-1} - 2u_j)}{h^2} + O(h^2) = \tilde{D}_2 u + O(h^2). \tag{7.5}$$

We can now use $\tilde{D}_2$ to rewrite (7.4) as

$$D_2 u = u_x + \frac{h^2}{6}\tilde{D}_2 u_x + O(h^4). \tag{7.6}$$

We can rewrite (7.6) in matrix notation as

$$D_2 u = (I + \frac{h^2}{6}\tilde{D}_2)u_x + O(h^4). \tag{7.7}$$

where I is the identity matrix and we can identify $\tilde{D}_2$ as a tridiagonal matrix (see (7.5)).

If you neglect boundaries you see that you can now invert this matrix to express u_x in terms of u

$$u_x = (I + \frac{h^2}{6}\tilde{D}_2)^{-1} D_2 u + O(h^4). \tag{7.8}$$

Equation (7.8) says that you can get a fourth order accurate approximation to u_x using only a three point stencil, but you have to solve a tridiagonal system of equations to get u_x from the function values u. This is the compact implicit approximation.

Typically compact implicit spatial differencing is coupled with Runge-Kutta time integration.

Consider the equation

$$u_t = u_x,$$

and as before let v denote the numerical approximation. If we consider first a semi-discrete approximation (i.e., do not yet discretize in time) then we have the system of ordinary differential equations

$$dv_j/dt = (I + \frac{h^2}{6}\tilde{D}_2)^{-1} D_2 v. \tag{7.9}$$

Note that on the right hand side all of the values of v_j are coupled together. (The inverse of a tridiagonal matrix is a full matrix.) Now you can apply any time integrator you want to (7.9). One common scheme is Runge-Kutta. You do not explicitly compute the inverse of

$$(I + \frac{h^2}{6}\tilde{D}_2).$$

Rather you write the update equations for v_j and then solve a tridiagonal scheme at each stage. In order to see how this works suppose we consider forward Euler. Note, that this is essentially what you would do for <u>each</u> stage of Runge-Kutta. Letting v_j^n denote the numerical solution we have

$$(I + \frac{h^2}{6}\tilde{D}_2)(v_j^{n+1} - v_j^n) = \Delta t D_2 v. \tag{7.10}$$

This gives a tridiagonal matrix that has to be solved at level $n+1$. Such a system would have to be solved at each stage of Runge-Kutta. The result is a 4-4 scheme.

Note that if you use Runge-Kutta you will also not get unconditional stability - there will still be a stability limit even though you are doing the work of an implicit scheme. Generally the point of the compact implicit schemes is to get higher accuracy without the boundary problems of increasing the stencil. Compact implicit methods require some boundary treatment to close the tridiagonal matrix (as do all implicit methods) and the fourth order accuracy and stability can be very sensitive to the boundary treatment. This approach has also been extended to yet higher order schemes (e.g., there are sixth order compact implicit schemes).

Implicit Schemes for Systems

There is one more issue that you should consider when using implicit schemes. Suppose instead of a scalar equation you consider an $m \times m$ system of equations

$$\vec{u}_t = A\vec{u}_x. \tag{7.11}$$

You can use any of the implicit schemes that we have discussed for (7.11). Consider for concreteness Crank-Nicolson. However, there is one <u>big</u> difference between the system

case and the scalar case. In the scalar case these systems give rise to tridiagonal matrices. It is very easy and efficient to solve tridiagonal systems of equations. In fact if you have n points, tridiagonal systems can be solved in $O(5n)$ operations. There is also a lot of free software to solve tridiagonal systems of equations so you should never have to write your own program.

In the system case you get a block tridiagonal system of equations. That is you get a system of equations that is tridiagonal in structure if you look at blocks of size m, the size of the system. In this case the solution process is still easy and still proportional to the number of points n, however you must invert each block. The cost of doing such an inversion is $O(m^3)$ as each block is typically full and the cost of Gaussian elimination on a full $m \times m$ matrix is $O(m^3)$. Thus the cost of inverting the block tridiagonal matrix for an implicit scheme for systems is now $O(m^3 n)$. Generally you think of n as a number that can be large while m is typically $O(1)$ and is fixed by the problem. This is a reasonable way to think of things when you consider the asymptotic limit $h \to 0$. However, in practice increasing m can have a dramatic effect on the cost (computer time) of a computation. For example, if m increases from 2 to 3, m^3 increases from 8 to 27, more than a threefold increase.

Thus you should be aware that the cost of an implicit scheme for systems rises rapidly as the size of the system increases. In some special cases there is a special structure to the blocks that you can take advantage of when you invert the blocks, but generally you have a full block which must be inverted at each grid point.

8 SEMI-IMPLICIT SCHEMES

In some problems you may want to be implicit on some terms but not on others. Consider the equation

$$u_t = u_x + R(u), \tag{8.1}$$

where $R(u)$ is a given nonlinear function. This models what happens in chemistry for example, where there are nonlinear chemical reaction terms. Thus (8.1) is a model of an advection-reaction equation (the u_x term models the advection of disturbances while the $R(u)$ term models the reaction.)

Suppose you wanted to be implicit on the whole equation (8.1). For simplicity, suppose you wanted to use Crank-Nicolson. In this case at each time step you would have a system of nonlinear equations to solve. This can be enormously more expensive and difficult to solve than the system of linear equations you get without the nonlinear reaction term. In order to solve a nonlinear system of equations you would typically use Newton's method for systems. If you wrote the system as

$$\vec{F}(\vec{u}) = 0,$$

the analogue of Newton's method for systems is

$$\vec{u}_{n+1} = \vec{u}_n - J(\vec{u}_n)^{-1}\vec{F}(\vec{u}_n),$$

where J is the Jacobian matrix for $\vec{F}$ (i.e, the matrix of all partial derivatives of the vector function $\vec{F}$). Thus, in order to appy Newton's method for systems you must compute J at each iteration and then compute its inverse. While there are approaches to reduce some of the computations, this is typically very computationally intensive.

One approach to this class of problems is to use semi-implicit schemes. These schemes are implicit on some terms (for example the linear term in (8.1)) and explicit on other terms (in this case the nonlinear reaction term). The most common semi-implicit scheme is the Crank-Nicolson Adams-Bashforth scheme, which combines the implicit Crank-Nicolson scheme with the explicit second order Adams-Bashforth scheme. The Adams-Bashforth family of schemes is a family of explicit schemes for ordinary differential equations that uses previous timesteps to update to the new timestep.

The scheme is

$$\frac{(v_j^{n+1} - v_j^n)}{\Delta t} = \frac{1}{2}\frac{(v_{j+1}^{n+1} - v_{j-1}^{n+1} + v_{j+1}^n - v_{j-1}^n)}{2h} + \frac{3}{2}R(v_j^n) - \frac{1}{2}R(v_j^{n-1}). \tag{8.2}$$

The explicit treatment is exactly the second order Adams Bashforth scheme. Those of you who have studied numerical methods for ordinary differential equations may be aware of it. You can see that it is second order by simple manipulation of Taylor series.

You can ask why this choice for the explicit component of a semi-implicit scheme. In fact second order Adams Bashforth has a relatively large truncation error due to the wide stencil in time (i.e., time level $n-1$) and the highly uncentered nature of the scheme. However, in a semi-implicit scheme the major expense at each timestep is due to the solution of the equations (in this case linear) that you get from the implicit scheme. It is important that the explicit component involve as few stages as possible so as to minimize the number of times you have to solve these equations. Thus for efficiency considerations Adams Bashforth is generally preferred to a predictor corrector type scheme (for example like Runge-Kutta) which would require 2 solutions of the implicit equations for each timestep. However, the use of a more centered explicit scheme can give more accuracy.

By now you should be able to think of other variations of semi-implicit schemes. For example you can use the general family of schemes defined in (7.3) where the parameter α is a free parameter. Furthermore, the implicit equations should generally be solved using the $\delta-$formulation. An alternative to semi-implicit schemes is operator splitting which we will discuss in the context of 2 dimensional problems.

9 PARABOLIC PROBLEMS

We consider the basic heat equation

$$u_t = au_{xx}, \tag{9.1}$$

where we have explicitly indicated the diffusivity a. Many of the schemes that we have discussed previously can be applied to (9.1). However, you should realize that leap frog is not one of them. (See the discussion of artificial dissipation and the analysis leading up to (5.31).)

However, there is a problem with explicit schemes applied to parabolic equations. This is that stability often requires a timestep restriction of the form

$$\Delta t/h^2 \leq w_0. \tag{9.2}$$

Equation (9.2) says that if you cut h in half, the maximum stable timestep decreases by a factor of four. Thus explicit schemes are very restrictive in terms of timestep.

Before we prove this, you should suspect some restriction of the form (9.2) simply on the grounds of dimensional scaling. Consider a hyperbolic equation,

$$u_t = cu_x.$$

In this equation c has units of speed (length/time). Thus $\lambda = c\Delta t/h$ is nondimensional and it is reasonable to get a stability bound of the form $\lambda \leq w_0$. Examination of (9.1) indicates that a has unites of length^2/time. Thus $a\Delta t/h$ is not nondimensional and can not be obtained from a stability analysis. (In contrast $a\Delta t/h^2$ is nondimensional and is in fact what you typically get as a stability restriction for explicit schemes.)

In order to illustrate this, consider the forward Euler scheme applied to (9.1),

$$v_j^{n+1} = v_j^n + a\frac{\Delta t}{h^2}(v_{j+1}^n + v_{j-1}^n - 2v_j^n). \tag{9.3}$$

We can do the von Neumannn analysis exactly as we did before. Set

$$v_j^n = z^n \exp(ikhj),$$

and plug into (9.3). We get

$$z = 1 - 2a\frac{\Delta t}{h^2}(1 - \cos(kh)). \tag{9.4}$$

Note that the main difference between the second derivative and the first derivative case is that for u_x the central difference approximation gives you terms like $i\sin(kh)$ while for the second derivative you get terms like $1 - \cos(kh)$. It follows from (9.4)

that we always have $z \leq 1$. Thus in order to get stability ($\mid z \mid \leq 1$) we must find conditions such that

$$z \geq -1.$$

For any value of kh this implies

$$a\frac{\Delta t}{h^2} \leq \frac{1}{1-\cos(kh)}. \tag{9.5}$$

In order to have stability, you must have stability for all kh. This means that you must make the denominator on the righthand side of (9.5) as big as possible, namely set $kh = \pi$. This gives the stability bound for forward Euler as

$$\Delta t \leq \frac{h^2}{2a}. \tag{9.6}$$

For other explicit schemes the constants will vary, but the scaling of Δt with h^2 will remain the same.

This severe timestep restriction makes explicit schemes very inefficient for most parabolic equations. Instead implicit and semi-implicit schemes are used. The implicit schemes include backward Euler, Crank-Nicolson and the general scheme defined in (7.3). Another thing that you should be aware of is that implicit schemes for parabolic equations generally lead to much better conditioned matrices than for hyperbolic equations. For example, suppose you were doing backward Euler on (9.1). If we set

$$\beta = a\frac{\Delta t}{h^2},$$

we get the following system of equations

$$(1+2\beta)v_j^{n+1} - \beta v_{j+1}^{n+1} - \beta v_{j-1}^{n+1} = v_j^n. \tag{9.7}$$

Notice that the effect of the second derivative is to make the coefficient of v_j^{n+1} (which is also the diagonal of the matrix) larger. Said another way the matrix becomes more diagonally dominant. (A matrix is said to be diagonally dominant if for each row the absolute value of the diagonal element is greater than the sum of the absolute value of all of the off diagonal elements in that row.) Thus diagonally dominant matrices look (and often act) like diagonal matrices. It is generally the case that the more diagonally dominant a matrix is (i.e., the bigger the diagonals become relative to the off diagonals) the better conditioned the matrix becomes, i.e., there is less chance of numerical errors in performing the Gaussian elimination. In addition, such matrices are typically more amenable to iterative methods of solution. Observe that in contrast when you have u_x on the righthand side and do central differencing it does not contribute to making the diagonals bigger (see 5.14)).

Semi-implicit schemes are very common when you have reaction-diffusion or reaction-advection-diffusion equations. For example, suppose you had an equation of the form

$$u_t = u_{xx} + u_x + R(u). \tag{9.8}$$

At this point you should be able to identify the physical role of each term on the right-hand side, u_{xx} corresponds to diffusion or dissipation, u_x corresponds to advection, $R(u)$ corresponds to reaction. In many instances the parabolic timestep restriction is very limiting. Thus you would want to be implicit on the parabolic terms, but do not want to be implicit on the advection or reaction terms. In this case you would use a semi-implicit scheme, Crank-Nicolson on the diffusion terms, second order Adams-Bashforth on all of the rest of the terms. Note that such a scheme would not be unconditionally stable, but it would not have the severe parabolic stability bound.

One problem in solving parabolic problems with implicit schemes is that you have to experiment with the timestep. You will be using a scheme that is unconditionally stable, but which could be very inaccurate if the timestep is too large. Thus you may have to experiment until you get an accurate Δt. In order to illustrate what can happen if you are not sufficiently careful with the timestep, we will discuss one such scheme, namely the Dufort-Frankel scheme.

Consider the simple heat equation

$$u_t = u_{xx},$$

with leap frog time differencing,

$$v_j^{n+1} - v_j^{n-1} = \frac{2\Delta t}{h^2}(v_{j+1}^n + v_{j-1}^n - 2v_j^n). \tag{9.9}$$

You should know by now that this scheme is unstable (see the argument leading up to (5.31)). In the Dufort-Frankel scheme (9.9) is stabilized by taking the v_j^n term on the righthand side and averaging at levels $n+1$ and $n-1$. The resulting scheme is

$$v_j^{n+1} - v_j^{n-1} = 2\frac{\Delta t}{h^2}(v_{j+1}^n + v_{j-1}^n) - 2\frac{\Delta t}{h^2}(v_j^{n+1} + v_j^{n-1})). \tag{9.10}$$

Note that (9.10) is implicit but in a very simple way as you can still solve directly for v_j^{n+1}. If we rewrite the scheme we get

$$(1 + 2\frac{\Delta t}{h^2})v_j^{n+1} = (1 - 2\frac{\Delta t}{h^2})v_j^{n-1} + 2\frac{\Delta t}{h^2}(v_{j+1}^n + v_{j-1}^n). \tag{9.11}$$

You can see that the coefficient of v_j^{n+1} becomes larger due to the implicit scheme. Thus you make things smaller when you solve for v_j^{n+1}. This is always a good sign that you are stabilizing things. Another way to look at it is that the matrix (now diagonal) becomes even more diagonally dominant if you increase Δt. We will not do the von Neumann analysis, but it is easy to do and to find that (9.10) is unconditionally stable.

You may think this is great, because you now get stability with no matrix to invert. But suppose we do a simple rearrangement of (9.11). We can write

$$v_j^{n+1} - v_j^{n-1} = 2\frac{\Delta t}{h^2}(v_{j+1}^n + v_j^n - 2v_j^n) - 2\frac{\Delta t}{h^2}(v_j^{n+1} + v_j^{n-1} - 2v_j^n). \tag{9.12}$$

Now suppose we manipulate the last term on the righthand side of (9.12) to get

$$\frac{v_j^{n+1} - v_j^{n-1}}{2\Delta t} = \frac{v_{j+1}^n + v_{j-1}^n - 2v_j^n}{h^2} - (\frac{\Delta t}{h})^2 \frac{(v_j^{n+1} + v_j^{n-1} - 2v_j^n)}{\Delta t^2}. \tag{9.13}$$

Now suppose you take advantage of the large timesteps to choose $\Delta t \to 0$ and $h \to 0$ in such a way that $\lambda = \Delta t/h$ is constant. The lefthand side of 9.13) converges to v_t. The first term on the righthand side converges to v_{xx} which is what we have in the original equation. However, look at the second term in the righthand side of (9.13). As $\Delta t \to 0$ this term converges to $-\lambda^2 v_{tt}$. Thus the limiting partial differential equation under the condition $\Delta t/h = \lambda$ is

$$v_t = v_{xx} - \lambda v_{tt},$$

which is the wrong equation. If of course you took $\Delta t \to 0$ and $h \to 0$ in such a way that $\beta = \Delta t/h^2$ is constant you would get the right equation.

In practical computations you may have to do a lot of experimentation with the timestep. If you are interested only in stationary steady states (i.e., solutions which for large time are independent of t), then you probably can get away with large timesteps. If you are interested in the transient you will have to be more careful. You will often not know in advance the timescale on which the solution varies. This is particularly true for nonlinear problems. Remember that in practice you do not let $\Delta t \to 0$ and $h \to 0$. Rather you make only a few computations and verify that the numerical solutions are converging to something. In this case you will have to be very careful in how you access the accuracy of your solution, particularly in regards to errors in time.

Boundary Conditions

We next consider appropriate boundary conditions for parabolic problems. Generally, the numerical treatment of boundary conditions for parabolic problems is much easier than for hyperbolic problems so we deal with parabolic problems first. Consider the heat equation on a bounded interval, for example

$$u_t = u_{xx} \quad 0 \le x \le 1.$$

Generally you expect to have to impose 2 boundary conditions. You can see why by considering what happens at steady state, i.e., if $u_t = 0$. In this case you have the second order equation $u_{xx} = 0$ which needs 2 boundary conditions. For parabolic problems you typically impose one boundary condition at each endpoint.

You can impose any combination of u and u_x. The most general representation of the boundary conditions is

$$\begin{aligned} \alpha u(0,t) + \beta u_x(0,t) &= f_0(t) \\ \gamma u(1,t) + \delta u_x(1,t) &= f_1(t) \end{aligned} \tag{9.14}$$

Boundary conditions of the general form (9.14) are called impedance boundary conditions. In the special case $\beta(\delta) = 0$ the boundary condition is called a Dirichlet boundary condition, while if $\alpha(\gamma) = 0$ the boundary condition is called a Neumann boundary condition.

Dirichlet boundary conditions can be implemented easily in implicit schemes, simply by using the boundary condition as the equation for a boundary point. For example if you have a Dirichlet boundary condition and you use the $\delta-$formulation then the equation for the boundary point would simply be

$$\delta_0 = f_0(t + \Delta t) - f_0(t).$$

There are many ways to implement boundary conditions involving derivatives. One of the most common and most robust is by using what are called fictitious points. Suppose that at $x = 0$ you want to impose the boundary condition

$$u_x(0, t) = 0.$$

This boundary condition can be implemented by adding a fictitious point at $j = -1$. When you add an additional point you need an additional equation. The additional equation is the boundary condition. To see how this works, suppose we use v_j^n to denote the numerical solution. Suppose further that we were using Backward Euler. Then at $j = 0$ we would have two equations,

$$\begin{aligned} v_0^{n+1} &= v_0^n + \Delta t \frac{v_1^{n+1} + v_{-1}^{n+1} - 2v_0^{n+1}}{h^2}, \\ \frac{v_1^{n+1} - v_{-1}^{n+1}}{2h} &= 0. \end{aligned}$$

The case of general impedance conditions can be handled similarly.

10 INITIAL BOUNDARY VALUE PROBLEMS

We next consider the effect of boundaries for hyperbolic problems. For simplicity we first consider problems defined on a <u>half</u> open interval (e.g., problems defined on the interval $-\infty < x \leq 0$). Boundaries can introduce two complications to numerical methods.

1. Sometimes boundary conditions must be imposed as part of the given equation
2. Whether or not this is true, some numerical procedure must be imposed to deal with the fact that grid points to the left or right of the boundary are not available.

We refer to the first point as the problem of boundary conditions, while we refer to the second point as the problem of numerical boundary conditions.

We consider first just the problem of boundary conditions. Thus we consider PDEs without finite difference approximations.

Consider first the wave equation

$$u_t = u_x, \tag{10.1}$$

where the problem is defined on the interval

$$-\infty < x \leq 0.$$

We certainly need initial data at $t = 0$,

$$u(0, x) = f(x). \tag{10.2}$$

Should we impose a boundary condition at $x = 0$?

In order to understand this problem, it is convenient to consider (10.2) as a problem in the $x - t$ plane, with x as the horizontal axis and t as the vertical axis. Thus the boundary in the $x-t$ plane is the vertical axis $x = 0$. We know the exact solution to (10.1). The solution is

$$u(t, x) = f(t + x). \tag{10.3}$$

Thus initial data is carried along the characteristic lines

$$t + x = c,$$

where c is a constant (see the discussion on upwind schemes right after (4.22)).

Now you can ask whether the solution (10.3) is sufficient to determine the solution to the partial differential equation for <u>all</u> values of t. The answer is <u>no</u>. Suppose for example you look at values of t and x such that $x + t = 1$. The initial data $f(x)$ is not defined for this value of $t + x$. In fact from (10.2) you can only assume that $f(x)$ is defined for $x \leq 0$.

Thus in the presence of boundaries the characteristic curves emanating from the initial line ($t = 0$) can <u>not</u> determine the solution for all values of t. What happens is that the initial data determines the solution only up to the bounding characteristic curve $t + x = 0$. Beyond that the initial data does not determine the solution.

You can now ask how you would determine the solution at such a point. For concreteness take a specific point, $x = -2$, $t = 3$. Since you know that the solution is constant on the characteristic curve $x + t = 1$ you can continue this curve backwards (i.e., let x decrease). You can see that the characteristic curve intersects the boundary $x = 0$ before it hits the initial line $t = 0$. In order to get values for the solution along this characteristic curve we <u>must impose</u> a <u>boundary condition</u> along this line,

$$u(t, 0) = g(t). \tag{10.4}$$

This makes sense geometrically, but there is another way to see that you need a boundary condition that is more analytic in nature. Consider just (10.1-10.2). You can ask whether this problem is well posed or not. Let's look for solutions of the form

$$u(t,x) = \exp(\sigma t)\exp(\mu x) \;\; \mu > 0. \tag{10.5}$$

This may look similar to the Fourier analysis that we did for problems defined on the whole line, but there is a difference. Solutions with spatial dependence of the form $\exp(\mu x)$ are not allowed when you consider problems posed on the infinite interval $-\infty < x < \infty$, because they blow up as $x \to \infty$. However, now that we have only the half open interval $-\infty < x \leq 0$, these solutions have a perfectly reasonable spatial dependence and you would expect to have bounded solutions if the problem were well posed. Note that you do not expect Fourier type solutions, because you expect there to be a regularity condition at $-\infty$, e.g., the solution goes to 0 as $x \to -\infty$. Those of you who have encountered Laplace transforms should be familiar with this.

Suppose we substitute (10.5) into (10.1). We will find $\sigma = \mu > 0$. Thus if we do not impose any boundary condition, we will get solutions that are perfectly well behaved in x but explode exponentially as t increases, in fact solutions that grow as $\exp(\mu t)$ for any $\mu > 0$. Such a problem cannot be well posed.

Now suppose that you did impose a boundary condition of the form (10.4) with $g(t)$ bounded. Then clearly you can not get exponentially growing solutions of the form (10.5). Thus imposition of a boundary condition filters out the exponentially growing solutions and makes (10.1) well posed. Also observe that if you want your solution to be continuous $g(t)$ can not be arbitrary. There is a compatibility condition that must be satisfied at the origin of the $x - t$ plane,

$$g(0) = f(0),$$

If this condition is violated you still get solutions, however there will be a discontinuity across the bounding characteristic curve $t + x = 0$.

You should try to think geometrically about what this implies. The family of characteristic curves $x + t = c$ moves to the left as t increases. Thus the characteristic curves enter the computational domain $-\infty < x \leq 0$ as t increases. In this case you must impose a boundary condition in order for the problem to be well posed.

What if characteristic curves leave the computational domain as t increases? In order to consider this case consider the same equation (10.1) but now on the interval

$$0 \leq x < \infty.$$

In this case should you impose a boundary condition? The answer is no. The characteristics travel to the left. Thus in the $x - t$ plane every point on the t axis (the vertical line $x = 0$) can be reached by a characteristic propagating from the interior of the domain. Thus the solution at $x = 0$ is completely determined from the initial

condition (i.e., from the interior) and no boundary condition is needed nor should one be imposed.

You can also ask if you still have exponentially growing solutions. Instead of (10.5) you now have to consider solutions that decay as $x \to \infty$. Thus look for solutions of the form

$$u(t,x) = \exp(\sigma t)\exp(\mu x) \;\; \mu < 0. \tag{10.6}$$

From the equation (10.1) you now get $\sigma = \mu < 0$. Thus these solutions decay as t increases and do not signify that the problem is ill posed.

Of course you could argue that you want to consider solutions that grow like $\exp(\mu x)$ with $\mu > 0$. These solutions grow as $x \to \infty$ and also will grow exponentially in time. However, these solutions are excluded from the analysis. Usually when you consider well posedness it has to be considered in the context of a certain space of solutions. The natural space to consider is $L_2(0,\infty)$ or some other space where the solutions have some sort of regular behavior near $x = \infty$. Said another way, even though no boundary condition is imposed at ∞, you are assuming that the solutions satisfy certain regularity or integrability conditions near ∞.

Alternatively, suppose we consider the equation

$$u_t = -u_x, \tag{10.7}$$

defined on the interval

$$-\infty < x \leq 0,$$

with initial condition

$$u(0,x) = f(x). \tag{10.8}$$

Should we impose a boundary condition at $x = 0$? The solution to (10.7) is now constant on characteristic curves $t - x = c$, i.e., lines sloping at 45° in the plane. Since the characteristic curves slope to the right the solution at $x = 0$ is now completely determined from the solution in the interior. You do not need a boundary condition. To impose a condition would be overspecifying the solution. In fact, it could make things worse because there would be an incompatibility between the condition that you impose and the values that the solution 'wants" to take at $x = 0$. This can also be seen analytically by looking for exponential solutions as in (10.5). You now get $\sigma = -\mu$ so that these solutions decay in time and the problem is well posed without boundary conditions.

Boundary Conditions for Systems

Let's next look at the two way wave equation written as a first order system,

$$\begin{pmatrix} u \\ v \end{pmatrix}_t = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x, \quad -\infty < x \leq 0, \tag{10.9}$$

with initial conditions

$$u(0,x) = f_1(x), \quad v(0,x) = f_2(x), \quad -\infty < x \leq 0.$$

Now you know that the eigenvalues of the matrix in (10.9) are ± 1. This means that the characteristic speeds are ± 1, so that there are solutions that behave like $x+t$ and solutions that behave like $x-t$. By simple manipulation of the (10.9) it is easy to see that $u+v$ satisfies

$$(u+v)_t = (u+v)_x,$$

while $u-v$ satisfies

$$(u-v)_t = -(u-v)_x.$$

These are called the characteristic variables of (10.9).

What do you expect you will have to do for boundary conditions? By what we have done for the scalar case you should expect to have to impose 1 boundary condition at $x=0$ to account for the wave that travels on the leftward moving family of characteristic curves $x+t=c$. You can see this analytically. Look for solutions of the form

$$\begin{pmatrix} u \\ v \end{pmatrix} = \exp(\sigma t)\exp(\mu x)\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}, \tag{10.10}$$

with $\mu > 0$ (corresponding to solutions that decay exponentially as $x \to -\infty$). Plugging (10.10) into (10.9) we get

$$\sigma\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \mu\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}. \tag{10.11}$$

Since the eigenvalues of

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

are ± 1 it follows that

$$\sigma = \pm\mu$$

and that

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix},$$

must be an eigenvector of A. The "bad" solutions, i.e., those with $\sigma > 0$ correspond to the eigenvalue $+1$ of A, that is to the characteristic variable $u+v$ (the eigenvector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ of A).

Thus the the equation (10.9) by itself is ill posed. If, however, we impose the incoming characteristic variable $u+v$ at $x=0$ we rule out the exponentially growing solutions and the system becomes well posed. Ideally we would want to have a boundary condition of the form

$$u(t,0) + v(t,0) = g(t). \tag{10.12}$$

Boundary condition (10.12) will give a well posed problem and is natural from the point of view of the characteristic variables. However, it is not the most general

boundary condition that you can have. For example, suppose you had a boundary condition of the form

$$u(t,0) + v(t,0) = \gamma(u(t,0) - v(t,0)) + g(t). \tag{10.13}$$

Then we would express the incoming characteristic variable in terms of the outgoing characteristic variable and prescribed data. Note that by setting $\gamma = \pm 1$ we can impose either of the two unknowns u or v.

You can see that boundary conditions of the form (10.13) also lead to well posed problems by excluding the growing solutions. In fact any growing solution is of the form (10.10) with $\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}$ an eigenvector of A corresponding to the eigenvalue 1 (i.e., to a solution obtained by adding $u(t,x) + v(t,x)$) and exponential growth is ruled out for such solutions by the boundary condition (10.13). On the other hand suppose that you imposed a boundary condition on the "wrong" characteristic variable, $u(t,x) - v(t,x)$, corresponding to the eigenvalue -1. It is easy to see that you would now still get growing solutions, i.e., the problem would then be ill posed. Note that geometrically you can think of the characteristic variable $u(t,x) - v(t,x)$ as leaving the computational domain at $x = 0$ (propagating from left to right), while the characteristic variable $u(t,x) + v(t,x)$ enters the computational domain at $x = 0$ from the outside, so you need a boundary condition.

Thus you can see that boundary conditions play two (related) roles:

1. Prescribe the value of incoming characteristic variables, either directly or as some combination of outgoing characteristic variables and prescribed data.

2. Rule out solutions which grow exponentially in t with arbitrarily large growth rates, but which are well behaved in x.

General System

Now let's consider a general system of m equations,

$$\vec{u}_t = A\vec{u}_x, \quad -\infty < x \leq 0, \tag{10.14}$$

where A is an $m \times m$ matrix. How many conditions should you impose at $x = 0$? Let's suppose for simplicity that A does not have zero as an eigenvalue. Let A have r positive eigenvalues and $m - r$ negative eigenvalues. By what we have already done, the r positive eigenvalues correspond to incoming characteristic variables while the $m - r$ negative eigenvalues correspond to outgoing characteristic variables. Thus you should expect that you would have to impose r conditions corresponding to the incoming characteristic variables.

Now let $\vec{e}_1^T, \vec{e}_2^T \ldots \vec{e}_r^T$ be the left eigenvectors of A corresponding to the r positive eigenvalues. Let I be the matrix with rows $\vec{e}_1^T, \vec{e}_2^T \ldots \vec{e}_r^T$. Thus I is a matrix of size

$r \times m$. Note that I does not stand for the identity matrix. We are using this notation because I will be used to obtain the incoming characteristic variables. Similarly let $\vec{f}_1^T, \vec{f}_2^T \ldots \vec{f}_{m-r}^T$ be the $m-r$ left eigenvectors for the negative eigenvalues and let O (representing outgoing) be the corresponding $(m-r) \times m$ matrix. We now define

$$\vec{u}_I = I\vec{u}, \quad \vec{u}_O = O\vec{u},$$

which are exactly the incoming and outgoing characteristic variables. Ideally you would like a boundary condition which imposes exactly $\vec{u}_I$, the incoming characteristic variables. However, (10.14) will be well posed with any boundary condition of the form

$$\vec{u}_I = S\vec{u}_O + \vec{g}(t), \tag{10.15}$$

where S is an $r \times m-r$ matrix and $\vec{g}(t)$ is an $r-$vector which is bounded in t.

Equation (10.15) gives a prescription as to what to do at a single boundary point. In practice you solve equations such as (10.14) on a bounded domain, say $a \leq x \leq b$. In this case you have to look at the incoming and outgoing characteristics curves and variables at each boundary point. For example, the right boundary point $x = b$ is like $x = 0$ in the preceding example. The incoming characteristic variables correspond to the positive eigenvalues. Thus you must impose a boundary condition for each positive eigenvalue (i.e., r boundary conditions for the example above). At the left boundary $x = a$ the roles of the positive and negative eigenvalues are reversed and you must impose a boundary condition for each negative eigenvalue (i.e., $m-r$ boundary conditions for the example above).

In applications A need not be just a constant. For example, suppose A depends on x and t. Then you have to look at the frozen coefficient matrices $A(a,t)$ and $A(b,t)$. Note that it may very well be the case that the number of positive and negative eigenvalues (i.e., the number of required boundary conditions) may change in time.

Finally, consider a system of conservation laws

$$\vec{u}_t = \vec{f}(\vec{u})_x. \tag{10.16}$$

In this case you have to rewrite (10.16) using the chain rule,

$$\vec{u}_t = J(\vec{u})(\vec{u})_x,$$

where $J(\vec{u})$ is the Jacobian matrix. (In applications J will be evaluated for the current solution, $J(\vec{u}^n)$). For each boundary point you must count the number of positive and negative eigenvalues and impose boundary conditions accordingly. It is definitely possible that the required number of boundary conditions will change in time.

We next address the issue of what to expect if you do not follow the boundary condition rule described above completely. From what we showed if you have an incoming characteristic variable and do not impose a boundary conditions then you get exponentially growing solutions. On the other hand if there is an outgoing characteristic variable and you happen to (incorrectly) impose a boundary condition on that

variable the solution will be bounded but inaccurate. In practice the solution will not blow up but will be oscillatory near the boundary where the superfluous boundary condition is imposed. Thus as a general rule:

- Underspecification is unstable.
- Overspecification is stable but can lead to oscillations (and possibly nonlinear instabilities).

It is possible that in some circumstances (highly nonlinear problems which are prone to instabilities) overspecification can stabilize what would otherwise be an unstable computation.

Example - Linearized Euler Equations

As an example of this theory of boundary conditions we consider the linearized Euler equations in 1 dimension. The Euler equations govern the motion of an inviscid gas, i.e., a gas in which viscous effects are negligible. The Euler equations are nonlinear and can be derived from general principles of conservation of mass, momentum and energy. One important application of the Euler equations occurs when we have small disturbances superimposed on a spatially constant ambient state. In this case some of the disturbances are acoustic in nature. They represent what we hear as sound. In many computations involving the Euler equations in general, and acoustics in particular, boundary conditions play a crucial role.

We will use the variables ρ, u and p to stand for the density, $x-$velocity and pressure respectively. The Euler equations in non-conservation form are

$$\begin{aligned} \rho_t + u\rho_x + \rho u_x &= 0 \\ u_t + uu_x + \frac{1}{\rho}p_x &= 0 \\ p_t + up_x + \gamma p u_x &= 0. \end{aligned} \tag{10.17}$$

where γ is the ratio of the specific heat at constant pressure to the specific heat at constant volume ($\gamma = 1.4$ for air). You can identify the first equation in (10.17) with conservation of mass, the second equation with conservation of momentum and the third equation with conservation of energy. This identification would be more apparent if we wrote the system in conservation form, but for the purpose of linearizing the equations to describe acoustic waves it is preferable to consider the system in non-conservation form.

Now suppose there is a spatially constant ambient state ρ_0, u_0 and p_0. Further suppose that you assume that the solution to (10.17) can be written in the form

$$\rho = \rho_0 + \rho', \ u = u_0 + u', \ p = p_0 + p', \tag{10.18}$$

where we assume that the primed variables in (10.18 are so small that terms that are quadratic in them can be neglected. We then plug (10.18) into (10.17) and linearize to get the linearized Euler equations

$$\begin{aligned} \rho'_t &= -u_0\rho'_x - \rho_0 u'_x \\ u'_t &= -u_0 u'_x - \frac{1}{\rho_0}p'_x \\ p'_t &= -\gamma p_0 u'_x - u_0 p'_x. \end{aligned} \tag{10.19}$$

Note that we have used the fact that spatial derivatives of the ambient (mean) state vanish. The system (10.19) describes the propagation of sound in the ambient medium. If you assume that the disturbance is isentropic, then ρ' is proportional to p' you can derive a second order wave equation for p'.

Now drop the primes for simplicity, take $u_0 \geq 0$ and consider (10.19) on the finite interval $a \leq x \leq b$. The boundary point $x = a$ is often called an inflow boundary since the flow field is entering the computational domain at this point from the outside (i.e., $u_0 > 0$). How many boundary conditions should you impose at an inflow boundary? By what we have already done you impose boundary conditions according to the negative eigenvalues of the matrix on the righthand side of the equations, i.e., the matrix A where

$$A = -\begin{pmatrix} u_0 & \rho_0 & 0 \\ 0 & u_0 & \frac{1}{\rho_0} \\ 0 & \gamma p_0 & u_0 \end{pmatrix}. \tag{10.20}$$

After some fairly simple linear algebra you can find that the eigenvalues of A (characteristic speeds of the system) are

$$-u_0, \quad \pm c_0 - u_0,$$

where $c_0 = \sqrt{\gamma p_0/\rho_0}$ is the speed of sound in the ambient medium.

Now there are two cases to consider. If $u_0 > c_0$ the flow at inflow is called supersonic because the flow velocity is greater than the speed of sound. In this case all eigenvalues are negative and so you impose all three variables at the boundary. If $u_0 < c_0$ the inflow boundary is called subsonic. In this case you impose two boundary conditions corresponding to the incoming characteristic variables. By looking at the left eigenvectors of A you can see that the characteristic variable corresponding to the eigenvalue $-u_0$ is $c_0^2\rho - p$. This corresponds to an entropy disturbance. The characteristic variable corresponding to the eigenvalue $-(c_0 + u_0)$ is $p + \rho_0 c_0 u$ while the characteristic variable corresponding to the eigenvalue $-(-c_0 + u_0)$ is $p - \rho_0 c_0 u$. These correspond to acoustic disturbances. Thus for a subsonic inflow the acoustic disturbances propagate in opposite directions while for a supersonic inflow all acoustic disturbance propagate to the right (this is usually called the downstream direction).

At a subsonic inflow you would like to impose $c_0^2\rho - p$ and $p + \rho_0 c_0 u$. Often you don't know this and have to specify other variables such as ρ and u. Observe that you

would never specify both p and u. This would correspond to specifying the outgoing characteristic variable $p - \rho_0 c_0 u$.

The boundary at $x = b$ is called an outflow boundary because the flow is moving out. Here you have to impose according to the positive eigenvalues of A. If $u_0 > c_0$ this is called supersonic outflow and you do not specify any boundary condition. If $u_0 < c_0$ this boundary is called a subsonic outflow boundary and you must specify one boundary condition. What you would like to do is to specify the outgoing acoustic variable $p - \rho_0 c_0 u$.

The problem of a subsonic outflow boundary is a classic problem in fluid dynamics. Often you do not have enough information to impose any condition at a subsonic outflow boundary. In contrast you may know everything that happens at inflow and would like to impose 3 conditions. Mathematically this is not correct. There have been many attempts to come up with a subsonic outflow boundary condition. In one dimension it is not too difficult. You make the assumption that all disturbances radiate outward from the computational domain. Thus you expect no information to enter the domain from $+\infty$. Mathematically this translates into setting the incoming characteristic variable to be its value at $+\infty$, i.e.,

$$p - \rho_0 c_0 u = 0. \tag{10.21}$$

Equation (10.21) is also an example of what is commonly called a non-reflecting boundary condition or a radiation boundary condition. Suppose you are really given the problem on the interval $a \leq x < \infty$. Thus you have no outflow boundary. All you know is that no radiation is propagating inward from ∞ (i.e., there are no sources of noise for large x). In practice you can only solve on a finite domain. Thus you must introduce an artificial boundary at say $x = c > a$. You must come up with a boundary condition to impose. The natural boundary condition is to impose that there is no disturbance corresponding to the incoming characteristic variable.

Note that this is not always so easy to implement in practice. For example, you may be solving the full Euler equations, not the linearized equations and you may not know exactly what to take for the ambient flow, e.g., p_0, u_0 and ρ_0 in order to define the disturbance. Said another way you may not know what to linearize the equations about. You may linearize about the previous timestep or make some estimate of what the ambient state should be. This entire theory is much more difficult in 2 and 3 dimensions. In this case you cannot readily get the incoming characteristic variable and you have to do a significant amount of modeling and analysis in order to get a reasonable boundary condition.

11 Numerical Treatment of Boundary Conditions

When you use finite difference schemes to approximate initial boundary value problems you have to face both the analytic issues of boundary conditions that we have described above together with some purely numerical issues.

In order to see what these issues are consider the equation

$$u_t = u_x, \quad 0 \leq x \leq 1, \tag{11.1}$$

combined with appropriate initial conditions. From what we have already seen no boundary condition is required at $x = 0$, while a boundary condition is required at $x = 1$. Now suppose that we approximate (11.1) by some finite difference scheme, for example leap frog

$$v_j^{n+1} = v_j^{n-1} + \lambda\,(v_{j+1}^n - v_{j-1}^n). \tag{11.2}$$

You should see directly from (11.2) that although no boundary condition need be imposed at $x = 0$ there is a numerical problem. The scheme (11.2) cannot directly be applied at $x = 0$. Suppose $x = 0$ corresponds to $j = 0$. Then in order to apply the scheme at $j = 0$ you will need a point at $j = -1$. You do not have such a point. This is an example of a problem that is purely numerical. The numerical scheme forces you to do some boundary treatment whereas no special treatment is required for the partial differential equation. Such problems require what is called a numerical boundary treatment.

Before we go on you might ask what causes this problem. This problem is directly related to the use of central differences. If you used one-sided differencing, i.e., a purely upwind scheme (see the discussion after (4.19)) then you would not need any special boundary treatment at $j = 0$. The upwind scheme follows the direction of propagation of information and requires no numerical treatment when a boundary condition is not applied. You might ask why we just don't go ahead and use upwind schemes. Such schemes are fairly straightforward for scalar equations. The problems arise when you try to use upwind schemes for systems of equations. In this case you can have waves propagating in different directions, i.e., you would have to use differencing in different directions for different components of the solution. Said another way, you would be forced to diagonalize the system into characteristic variables at each timestep and for each spatial point. Such a method could be very expensive and complicated. In most instances central differences are employed to avoid such a decoupling, however they cause numerical difficulties.

You can now ask what about $x = 1$. Since a boundary condition must be imposed at $x = 1$ you might think that there is no problem there. But suppose you were using the (2-4) leap frog,

$$v_j^{n+1} = v_j^{n-1} + \lambda\,(\frac{4}{3}\,(v_{j+1}^n - v_{j-1}^n) - \frac{1}{6}\,(v_{j+2}^n - v_{j-2}^n)). \tag{11.3}$$

Now think about applying (11.3) at the boundaries. Certainly at $x = 0$ ($j = 0$) there is a problem as with the (2-2) leap frog (11.2). However, there is also a problem at

the first grid point in from the boundary, $j = 1$. At $j = 1$ you need data at $j = 0$ and $j = -1$ in order to apply the scheme. Thus higher order schemes with wider stencils require numerical boundary treatments for more points than low order schemes with smaller stencils.

There is also a problem at the right boundary $x = 1$. Suppose that this point has index N. Then you will have given boundary data at $j = N$. However, in order to apply the scheme at $j = N - 1$ you need data at $j = N$ (this is given boundary data) and at $j = N + 1$ which you do not have. Thus higher order schemes also require numerical boundary treatments even at points for which a boundary condition is given.

There are 3 points to consider for numerical boundary treatments.

1. What about accuracy? Since you generally do central differencing in the interior but cannot do central differencing at the boundaries you might have difficulty maintaining accuracy at the boundary.

2. What about stability? You should now have many examples from the von Neumann analysis that perfectly reasonable treatments for the initial value problem can give rise to numerical instabilities. In the same way perfectly reasonable boundary treatments can give rise to instabilities for initial value problems. These can be analyzed in a manner similar to the von Neumann analysis, but the analysis is much harder.

3. In applications you will have two boundaries as in (11.1) above. How can you analyze the problem with two boundaries, in particular issues of accuracy and stability?

We can give an answer to point 3 immediately. For any given boundary treatment it can be shown that it is sufficient to consider each boundary treatment individually. That is for equation (11.1) above you only have to analyze problems on the intervals $0 \leq x < \infty$ and $-\infty < x \leq 1$. (The precise boundary points are irrelevant). In addition, if you have variable coefficients it is sufficient to consider only the problem with frozen coefficients.

We can also answer point 1 almost immediately. There is a general theorem which states that you can lose one order of accuracy at the boundary and still maintain the overall accuracy of the scheme. Specifically the theorem states that if you have a scheme of order (p, q) it is sufficient for the boundary treatment to be of order $(p - 1, q - 1)$ so that the overall accuracy is still of order (p, q). We will not prove this as it is mostly technical in nature. As an example of this theorem, if you had a second order scheme you would not want to use zeroth order extrapolation (i.e., extrapolating the boundary data from the interior)

$$v_0^{n+1} = v_1^{n+1}, \tag{11.4}$$

as (11.4) is only zeroth order accurate (the error is $O(h)$ but remember in analyzing truncation errors you always factor out one power of h). Rather you would use first order extrapolation

$$v_0^{n+1} = 2v_1^{n+1} - v_2^{n+1}, \tag{11.5}$$

which has a truncation error $O(h)$ and so is first order accurate. (Note you can keep track of the order of accuracy by seeing the degree of a polynomial for which an extrapolation formula is exact. (11.4) is exact only for constants (zeroth order accurate) while (11.5) is exact for linear functions (first order accurate). Remember that there are also stability issues. We will see that (11.5) is fine for Lax Wendroff but is unstable for leap frog. If you had a (2,4) scheme and you wanted to maintain the overall accuracy you would have to use third order extrapolation.

We next discuss the issue of stability. This theory is associated with a Swedish mathematician, Heinz Otto Kreiss (although others worked on it as well and the foundations were set prior to his work). We will give an overview of the Kreiss stability theory for initial boundary value problems. This theory is complicated and technical in nature and we will only give an outline. Consider for simplicity leap frog

$$v_j^{n+1} = v_j^{n-1} + \lambda\ (v_{j+1}^n - v_{j-1}^n). \tag{11.6}$$

as an approximation to

$$u_t = u_x, \quad x \geq 0. \tag{11.7}$$

Note that we are now looking only at the one boundary problem. We are using the property that it is sufficient to treat stability for each boundary independently. Let's first restrict to $\lambda \leq 1$ so that the basic scheme is von Neumann stable, i.e., we have stability for the initial value problem without any boundaries.

We want to follow the PDE analysis as closely as possible. Remember that when we analyzed just the partial differential equations we saw that in addition to the Fourier type solutions (i.e., solutions to (11.7)) that looked like $\exp(ikx)$, when we had a boundary value problem we also had to consider new types of solutions which decayed at ∞. When we have the interval $0 \leq x < \infty$ as in (11.7) this means solutions that look like

$$u(t,x) = \exp(\sigma t)\exp(\mu x), \quad \mu < 0.$$

You should look at the analysis that we did in (10.5) but remember there we considered the interval $-\infty < x \leq 0$ so that μ was positive.

In the PDE analysis we said that $\exp(\mu x)$ was a perfectly okay spatial dependence for the solution and thus it was important that there be no such solutions with $\sigma > 0$, otherwise the problem would not be well posed. On the finite difference level the analogue is to set

$$v_j^n = z^n \kappa^j, \quad |\ \kappa\ | < 1. \tag{11.8}$$

This is similar to the von Neumann analysis for the initial value problem except that then κ was replaced by $\exp(ikh)$, i.e., a complex number of magnitude 1.

Before we do any analysis you should see the analogue between the analytic solution and the discrete solution (11.8). In both cases z plays the role of $\exp(\sigma\Delta t)$. Solutions with $z > 1$ are unstable, they correspond to instabilities. Thus a stable computation must rule out such solutions. The spatial dependence $\exp(ikh)$ for the von Neumann analysis corresponds to Fourier analysis. Similarly for boundary value problems κ corresponds to $\exp(\mu)$ for $\mu < 0$.

Certainly one condition for stability of a boundary treatment is that the total scheme, with the boundary condition, admit no such solution. Just as for the PDE σ and μ are not independent. There is a relation between them, which is determined by the scheme (11.6). In order to determine this relationship plug (11.8) into (11.6). We get

$$(z^2 - 1)\kappa = \lambda z(\kappa^2 - 1). \tag{11.9}$$

Now when we did the von-Neumann analysis for the initial value problem all we had to do is show that for every κ on the unit circle there were no values of z with $\mid z \mid > 1$. However, for the initial-boundary value problem the situation is harder. Bad solutions with $z > 1$ and $\kappa < 1$ do exist. What we have to do is find these solutions and show that they are ruled out by the boundary conditions. We first solve (11.9) for κ as a function of z and then show how the boundary condition rules out such solutions.

First get a quadratic equation for κ in terms of z,

$$\kappa^2 + \frac{(1 - z^2)}{\lambda z}\kappa - 1 = 0. \tag{11.10}$$

Now you should interpret (11.10) as defining for each z with $\mid z \mid > 1$ two values of κ which we refer to as $\kappa_1(z)$ and $\kappa_2(z)$. We can take

$$\mid \kappa_1(z) \mid < 1, \quad \mid \kappa_2(z) \mid > 1. \tag{11.11}$$

You can ask why this is so. First, as for any quadratic the product of the roots has to be the last term in the quadratic, i.e., $\kappa_1(z)\kappa_2(z) = -1$. This says that either one root is inside the unit circle and one root is outside as in (11.11) or both roots are on the unit circle. Second, <u>none</u> of the κ's can be on the unit circle. Thus we can not have $\mid \kappa_i \mid = 1 \quad i = 1, 2$. In order to see why, remember that the problem is von Neumann stable. That means that means that if you set $v_j^n = z^n \exp(ikhj)$ and try to satisfy the scheme, there are <u>no</u> solutions with $\mid z \mid > 1$. Now suppose we just called $\exp(ikh)$ κ. Then κ is on the unit circle. From the von Neumann analysis, for any z κ and z will satisfy the quadratic (11.9). Furthermore, you know from von Neumann stability that $\mid z \mid$ <u>cannot</u> be greater than 1 with $\mid \kappa \mid = 1$. Thus when $\mid z \mid > 1$ neither of the two roots of (11.9) can be on the unit circle.

Now the solutions involving $\kappa_2(z)$ do not bother us because these solutions are <u>not</u> well behaved in x. They do not decay, but grow exponentially as $j(x) \to \infty$. However, the solutions with $\kappa_1(z)$ are a big problem. They are well behaved in x (j) but grow exponentially in time.

Now you may think this is very bad. However, remember what happened on the PDE level. If characteristics entered the region we had exponentially growing solutions which were ruled out by the boundary conditions. Now we have no entering characteristics, so there are no exponentially growing solutions for the PDE. However, there are exponentially growing solutions for the difference scheme. The finite difference method requires some boundary treatment. This is exactly what the numerical solution $z^n\kappa_1^j$ is telling us. The specification of a numerical boundary treatment should serve to rule out the exponentially growing solutions.

To be specific, let's suppose that we have a general boundary condition of the form

$$v_0^{n+1} = \sum_{j=0}^{q} c_j v_j^{n+1} + \sum_{j=0}^{q} d_j v_j^n. \tag{11.12}$$

(11.12) encompasses many numerical boundary treatments. For example, if we do extrapolation at time level $n+1$ we have

$$v_0^{n+1} = v_1^{n+1}, \tag{11.13}$$

while if we do one sided differencing we have

$$v_0^{n+1} = v_0^n + \lambda(v_1^n - v_0^n), \tag{11.14}$$

as you can easily see. What we have to make certain is that when you try to plug in the solution $z^n\kappa_1^j$ into any of the boundary treatments, no non-trivial solution results. For example, if you plug into (11.13) you get $\kappa = 1$ ruling out the bad solutions (but this is still unstable), while if you plug into (11.14) you get

$$z = 1 + \lambda(\kappa_1(z) - 1),$$

which has to be analyzed to see whether it allows the exponentially growing solutions.

We can now state a necessary but not sufficient condition for stability of a numerical boundary condition.

- Suppose a scheme is already von Neumann stable. If the boundary treatment is stable then (11.9) and (11.12) have no solutions with $\mid z \mid > 1$ and $\mid \kappa \mid < 1$. This is called the Ryabenkii-Godunov condition.

Now you should realize that the Ryabenkii-Godunov condition is hard to apply in practice. Let's compare it to von Neumann analysis. There in effect you plug in $z^n\kappa^j$ into the finite difference scheme alone, take $\mid \kappa \mid = 1$ and must show that the only values of z that satisfy the equation satisfy $\mid z \mid \leq 1$. Now you must not only do the von Neumann analysis but you must also solve (11.9) for $\kappa_1(z)$ and then plug into the boundary condition and verify that for all z with $\mid z \mid > 1$ there is no solution. Furthermore, we have considered a scheme where there is only a 3 point stencil. If you consider something like the (2-4) leap frog where there is a 5 point

stencil then it is easy to see that the analogue of the quadratic equation (11.10) is a quartic equation and there are now two κ's inside the unit circle. In this case the boundary treatment must account for the possibility of linear combinations of these two κ's as solutions. The problem is that this is only a necessary condition it is still possible that the scheme can be unstable.

There is a general theory of stability for initial boundary value problems that gives necessary and sufficient conditions for stability. We will not give details, but the upshot of the theory is that for a general boundary conditions an extension of the Ryabenkii-Godunov condition is almost sufficient as well as being necessary for stability of a given boundary treatment. The main problem with stability arises with what are called generalized eigenvalues. These are solutions where $\mid z \mid, \mid \kappa \mid \rightarrow 1$. The analysis of whether a generalized eigenvalue can cause instability is very delicate and can be very involved.

What you can do in practice is just run a simple program with the given boundary treatment and see if it blows up or not. This can show up solutions which are not generalized eigenvalues, however instabilities due to generalized eigenvalues can be very slowly growing and be hard to detect in practice. Generally boundary instabilities are very pernicious for non-dissipative schemes such as leap frog. Dissipative schemes such as MacCormack are much more robust in the variety of boundary conditions that they can handle.

12 Numerical Boundary Treatments

General Considerations

We now consider some numerical treatments that are employed in practice. There are two main problems that we will consider

1. Developing numerical boundary treatments
2. Dealing with systems where there are positive and negative eigenvalues (incoming and outgoing characteristic variables).

First consider the problem,

$$u_t = u_x \quad 0 \leq x \leq \infty, \quad u(0,x) = f(x). \tag{12.1}$$

By what we have done previously no boundary condition should be imposed because the characteristics exit at the boundary $x = 0$. However, if you do central differencing some boundary treatment has to be employed. Such a boundary is called an outflow boundary.

In some ways the best treatment for (12.1) is upwinding, i.e., use a scheme which does upwind differencing for all points, e.g.,

$$v_j^{n+1} = v_j^n + \lambda(v_{j+1}^n - v_j^n). \tag{12.2}$$

It's easy to see that (12.2) requires no boundary condition at $x = 0$ ($j = 0$) and in fact preserves the correct direction of information flow. In addition although (12.2) is only a (1,1) scheme you can easily see that it is exact for (12.1) provided $\lambda = 1$. (12.2) is the most basic upwind scheme and is called the Courant-Isaacson-Rees scheme.

Although (12.2) has advantages for general problems it is not often used as an interior scheme except around shocks. There are several reasons for this. For non-linear problems and problems with variable coefficients you cannot easily identify the characteristic variables. For problems with two characteristic speeds you will have to have the effective $\lambda < 1$ for at least one of the waves and you will then get only first order accuracy. For smooth functions it is more accurate to use high order central difference approximations. Finally it is not clear how to extend this to two dimensions.

We next discuss general numerical boundary treatments. In general, there are two major classes of numerical boundary treatments. Methods based on one-sided differences (i.e., (12.2)) and methods based on extrapolation. We consider extrapolation methods first since a better understanding of one-sided differences can be had if you first understand the notion of extrapolation.

Extrapolation

One major class of numerical boundary conditions is based on extrapolation at time level $n+1$. Extrapolation is a way of determining the solution at a given point, using only data on one side of that point. For example, for zeroth order approximation we have

$$v_0^{n+1} = v_1^{n+1}. \tag{12.3}$$

Another way of looking at (12.3 is that we approximate the solution as a constant function near the boundary. It is easy to see that (12.3) has an $O(h)$ error. Remember that you have to divide out a factor of h in defining the truncation error. Thus (12.3) is only zeroth order accurate and so can spoil the accuracy of second order schemes. First order extrapolation

$$v_0^{n+1} = 2v_1^{n+1} - v_2^{n+1}, \tag{12.4}$$

approximates the solution by a linear function near the boundary. From Taylor series you can see that the error is $O(h^2)$ so that (12.4) preserves the accuracy of second order schemes. In general you can symbolically define r^{th} order extrapolation by the formula

$$(E - I)^{r+1} v_0^{n+1} = 0 \tag{12.5}$$

where E is the shift operator,

$$E v_j^{n+1} = v_{j+1}^{n+1}.$$

Another way to define extrapolation is as an approximation to the vanishing of certain derivatives. If we define the forward difference operator

$$D_+ v_j^{n+1} = \frac{v_{j+1}^{n+1} - v_j^{n+1}}{h},$$

then r^{th} order extrapolation can also be written as

$$h^{r+1} D_+^{r+1} v_0^{n+1} = 0, \tag{12.6}$$

so that zeroth order extrapolation is an approximation to the vanishing of the first derivative, second order approximation is an approximation to the vanishing of the second derivative etc. Even though you can treat extrapolation as an approximation to the vanishing of some high order derivatives, you should treat it as a numerical procedure, rather than an approximation to an analytically imposed boundary condition.

It can be shown that extrapolation of any order is stable when coupled to any dissipative interior scheme (e.g., Lax Wendroff, MacCormack). First order extrapolation preserves the second order accuracy of second order schemes. However, you should realize that this is accuracy in the L_2 norm and there will be some degradation in accuracy near the boundaries.

One-Sided Differences

(12.2) can be used as a boundary condition. It can be shown that this is stable for any dissipative interior scheme. One nice way to implement one-sided differences is via a procedure called extrapolation of the fluxes. Suppose you are solving (12.1) and at time level n you define a point outside the domain, i.e.,

$$v_{-1}^n = 2v_0^n - v_1^n. \tag{12.7}$$

We can interpret (12.7) as approximating a fictitious value outside the domain (grid point $j = -1$) by first order extrapolation at time level n from the interior. Suppose you were to implement Lax Wendroff for (12.1) at $j = 0$ using (12.7) to get the needed value at $j = -1$. After some very simple algebra you can see that you will get exactly (12.2). Thus one-sided differences is equivalent to extrapolation at level n.

This formulation is very useful in dealing with conservation laws. Suppose that you were solving the nonlinear conservation law,

$$u_t = f_x \tag{12.8}$$

where $f(u)$ is a given nonlinear flux function. The use of Lax Wendroff, or its relative MacCormack requires you to compute the fluxes f_j and take differences of these fluxes. You can handle the boundary simply by doing first order extrapolation of the fluxes,

$$f_{-1}^n = 2f_0^n - f_1^n, \tag{12.9}$$

and then use the interior scheme.

The use of (12.7) or (12.9) is stable for dissipative schemes, it is probably the best boundary treatment to use with MacCormack and greatly simplifies the programming. You should bear in mind though that it is equivalent to one-sided differencing.

We next discuss MacCormack and the treatment of intermediate values in more detail.

Intermediate Values

We next look at the problem of what to do if you have intermediate values. We consider this in the context of MacCormack. There are several cases that need to be dealt with. First consider the FB variant of MacCormack,

$$\hat{v}_j = v_j^n + \lambda(f(v_{j+1}^n) - f(v_j^n)), \tag{12.10}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \lambda(f(\hat{v}_j) - f(\hat{v}_{j-1}))). \tag{12.11}$$

Note that we are working with a general flux function $f(u)$. Now at the predictor stage no numerical treatment has to be done at $j = 0$. However, some boundary condition must be imposed at the corrector stage. A very good procedure is to extrapolate the predicted fluxes as in (12.9).

$$f(\hat{v}_{-1}) = 2f(\hat{v}_0) - f(\hat{v}_1). \tag{12.12}$$

Note that in (12.12) we do not obtain a value for $\hat{v}_{-1}$. The extrapolation is used only to obtain an approximation for the flux function at the fictitious point. Typically you do the predictor step in a do loop, then the flux extrapolation (12.12) and then the corrector step (12.11) in another do loop.

A different treatment is required for the BF variant

$$\hat{v}_j = v_j^n + \lambda(f(v_j^n) - f(v_{j-1}^n)), \tag{12.13}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \lambda(f(\hat{v}_{j+1}) - f(\hat{v}_j))). \tag{12.14}$$

In this case you need some boundary treatment before completing the predictor stage, but you need no boundary treatment at the corrector case. A good approach is to extrapolate the fluxes at the predictor stage, i.e.,

$$f(v_{-1}^n) = 2f(v_0^n) - f(v_1^n) \tag{12.15}$$

and then use the interior scheme. In this case you do the extrapolation before doing the do loop for the predictor. Note again that you never get v_{-1}^n, but rather only the flux function at the fictitious point $j = -1$.

An important case arises when you have to impose a boundary condition for the PDE. For example, suppose you had the problem

$$u_t = -u_x, \quad 0 \le x \le 1 \quad u(t, 0) = g(t), \tag{12.16}$$

and consider just the boundary treatment at $x = 0$. Suppose you did the FB variant, a forward predictor and a backwards corrector. In this case you do not have to apply (12.11) at $j = 0$. You have boundary data that you can impose.

However, in order to apply (12.11) at $j = 1$ you need a predicted value at $j = 0$, i.e., $\hat{v}_0$. You can get this value by applying the predictor (12.10) at $j = 0$. You can also get this value from the given data, i.e., $\hat{v}_0 = g(t^{n+1})$. In this case you can use either approach. Both are stable. In most instances it is best to use the boundary data whenever possible so you would probably get slightly better results by applying the boundary data in the predictor. However, this is not always convenient, and another approach is to use the scheme and extrapolation of the fluxes whenever you have to and apply the given boundary data only at the end of each timestep.

Note that if you did the BF variant it would not matter what you did at $j = 0$ in the predictor stage as the predicted value at $j = 0$ is only used for the corrector at $j = 0$ and this is overwritten by the given boundary data.

Note that for Runge-Kutta with second order spatial differencing the situation is similar, however there are not so many different cases to consider. Generally you can extrapolate the fluxes at each stage. If you are given boundary data you have a choice of whether to extrapolate or impose the given data at the intermediate stages. Imposition of the given data can give slightly better results, but it is often more convenient to only impose the given data after the timestep is completed.

For (2-4) schemes the boundary treatment is more complicated. In this case you need some special treatment at both $j = 0$ and $j = 1$ and even if you have a boundary condition from the PDE at $j = 0$ you still need some special treatment at $j = 1$. Also your boundary treatment should be third order in space. The treatment for the (2-4) leap frog is complicated and we will not discuss it further. We will discuss the (2-4) MacCormack.

(2-4) MacCormack

Consider the equation

$$u_t = f_x$$

and the FB variant

$$\hat{v}_j = v_j^n + \frac{\lambda}{6}(-f_{j+2}^n + 8f_{j+1}^n - 7f_j^n), \tag{12.17}$$

followed by

$$v_j^{n+1} = \frac{1}{2}(\hat{v}_j + v_j^n + \frac{\lambda}{6}(7f(\hat{v}_j) - 8f(\hat{v}_{j-1}) + f(\hat{v}_{j-2}))). \tag{12.18}$$

Suppose that $f'(v_0^n) > 0$ so that no boundary condition needs to be prescribed at $x = 0$. You can see from (12.17) and (12.18) that you need a numerical boundary treatment at both $j = 0$ and $j = 1$. The best approach that I know of is to do a third order extrapolation of the fluxes. This at $j = 0$ you use the predictor as is (your do

loop starts at $j = 0$, but you extrapolate the fluxes at the corrector level

$$\begin{aligned} f(\hat{v}_{-1}) &= 4f(\hat{v}_0) - 6f(\hat{v}_1) + 4f(\hat{v}_2) - f(\hat{v}_3) \\ f(\hat{v}_{-2}) &= 4f(\hat{v}^n_{-1}) - 6f(\hat{v}_0) + 4f(\hat{v}_1) - f(\hat{v}_2). \end{aligned} \tag{12.19}$$

Note that the order that you do the extrapolation matters. You must first extrapolate to $j = -1$ and then to $j = -2$.

If you do the BF variant, so you do a backwards predictor then you extrapolate at level n before beginning the predictor.

This procedure works well even if you have an inflow boundary, i.e., if you have prescribed boundary data. I get good results if I use the extrapolation of the fluxes whenever I have to and then just impose the boundary data at the end of the timestep, when I am ready to advance the complete solution to level $n + 1$. Said another way, I never use the given boundary data at the predicted step.

Finally, note that a similar technique works for Runge-Kutta.

Some Boundary Treatments for Leap Frog

Now consider leap frog,

$$v_j^{n+1} = v_j^{n-1} + \lambda(v_{j+1}^n - v_{j-1}^n). \tag{12.20}$$

It can be shown that extrapolation at a fixed time level of any order is unstable when combined with (12.20). This is a very delicate instability connected with values of κ approaching the unit circle from the outside. It can be shown that one-sided differences is stable for leap frog.

Other stable approximations involve staggering the space-time levels. One such boundary condition is

$$v_0^{n+1} = v_0^n + \lambda(v_1^n - \frac{1}{2}(v_0^{n+1} + v_0^{n-1})). \tag{12.21}$$

(12.21) is sometimes more accurate than (12.2). The fact that you have to average in time should be no surprise to you as you have to do something similar when you have dissipation with leap frog and upwind differencing can be dissipative.

Systems

The last topic we will consider on boundary conditions is to deal with systems, for example,

$$\vec{u}_t = A\vec{u}_x, \quad 0 \le x < \infty,$$

where $\vec{u}$ is an $m-$vector and A is an $m \times m$ matrix. There are 3 cases to consider. Two are easy.

1. All characteristic curves enter at $x = 0$. In this case all of the eigenvalues of A are negative we need m boundary conditions. If you have extended stencils then you have to do something like one-sided differencing or extrapolation for points adjacent to the boundary.

2. All characteristics leave. In this case all eigenvalues of A are positive and you have to do some numerical boundary treatment for all variables.

3. Some characteristics enter and some characteristics leave. This is the hard case. Sometimes this is called an inflow/outflow boundary.

We consider an inflow/outflow boundary. All of the boundary conditions that we have discussed previously are only for the scalar equation. Now in our discussion of the initial value problem (without boundaries) we said that a system could be treated by being diagonalized and working only with the characteristic variables. Generally this does not work for initial boundary value problems because the unknowns are all coupled together by the boundary conditions.

We describe the problems of inflow/outflow boundaries by considering the 2-way wave equation,

$$\begin{pmatrix} u \\ v \end{pmatrix}_t = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x, \quad 0 \le x < \infty, \tag{12.22}$$

with appropriate initial conditions. Now you should know by now that the eigenvalues of the matrix in (12.22) are ± 1. Thus we need <u>one</u> boundary condition for the <u>one</u> negative eigenvalue (incoming characteristic variable). The incoming characteristic variable is $u - v$.

If everything was ideal you would be given prescribed data for $u - v$. However, generally you will not be given exactly the characteristic variable. A typical boundary condition will be to give one of the dependent variables, say

$$v(t, 0) = g(t). \tag{12.23}$$

Now we can write (12.23) in the form

$$u(t, 0) - v(t, 0) = u(t, 0) + v(t, 0) - 2g(t), \tag{12.24}$$

and this fits the general form we need for a well posed problem (the incoming characteristic variable is expressed as a combination of the outgoing characteristic variable plus given data - see (10.15)).

Now consider a numerical boundary treatment. In order to emphasize the theory we will first show how <u>not</u> to do it. Suppose you are using leap frog. You could say that at the boundary you know $v_0^{n+1} = g(t^{n+1})$ and since one-sided differencing is stable for leap frog (for the scalar equation) you could consider doing a one-sided differencing for u,

$$u_0^{n+1} = u_0^n + \lambda(v_1^n - v_0^n). \tag{12.25}$$

This gives you an explicit procedure to update u and v at the boundary. However, this will probably <u>not</u> work. It will probably be <u>unstable</u>. Remember, this is how <u>not</u> to do it.

You can ask why this probably will not work. We have

$$u = \frac{u+v}{2} + \frac{u-v}{2}.$$

Thus doing one-sided differencing for u is the same as doing one-sided differencing for a combination of the outgoing characteristic variable $u+v$ (which is stable and okay) and the incoming characteristic variable $u-v$ (which is unstable and goes against the flow of information). This will often make the whole procedure unstable.

Now we address the question of how you should do it. Suppose you wrote the equation for the outgoing variable

$$(u+v)_t = (u+v)_x, \tag{12.26}$$

and then did one-sided differencing for (12.26)

$$(u+v)_0^{n+1} = (u+v)_0^n + \lambda((u+v)_1^n - (u+v)_0^n). \tag{12.27}$$

(12.27) is now stable and gives a predicted value (or you can think of this as a tentative value) for $u+v$ at level $n+1$. Let's write this predicted value as $(\hat{u}+\hat{v})_0^{n+1}$. We are using this notation because this does not give the final values of u and v at the boundary - we also have to use the given boundary data. We do this by solving the system of linear equations

$$\begin{aligned} v_0^{n+1} &= g(t^{n+1}) \\ u_0^{n+1} + v_0^{n+1} &= (\hat{u}+\hat{v})_0^{n+1}, \end{aligned} \tag{12.28}$$

which can easily be solved for u_0^{n+1},

$$u_0^{n+1} = (\hat{u}+\hat{v})_0^{n+1} - g(t^{n+1}). \tag{12.29}$$

This procedure is called the characteristic boundary treatment and will be stable if the treatment of the scalar equation is stable.

Let us summarize how you would implement this treatment in practice. Suppose you use your numerical treatment to get tentative values for all variables. You can then form predicted values for the outgoing characteristic variables. You then write down a system of m equations using the given boundary data and the predicted values of the outgoing characteristic variables which you then solve at each timestep. If you have a nonlinear system then to get the characteristic variables you linearize around the previous timestep or sometimes around a given ambient state. This approach should always be used with inflow/outflow boundaries.

For the MacCormack or Lax-Wendroff schemes you typically do one-sided differentiation by extrapolating the fluxes as described above. Note, for the 2-4 MacCormack you have to extrapolate to two fictitious points. What is typically done is to update all variables using the extrapolated fluxes. Then at level $n+1$ you use these tentative

values to compute the outgoing characteristic variables. You then update all variables by solving a system of linear equations which says that the outgoing characteristic variables are equal to the tentative characteristic variables while the remaining equations are obtained from the boundary data. Assuming no zero eigenvalues the number of equations will be exactly the same as the number of variables. For nonlinear problems you do the same thing, but you have to approximate the characteristic variables using the Jacobian matrix.

13 Two-Dimensional Problems

Consider the 2D vector system

$$\vec{u}_t = A\vec{u}_x + B\vec{u}_y, \tag{13.1}$$

where $\vec{u}$ is an $m-$vector and A and B are $m \times m$ matrices. Equation (13.1) is hyperbolic if for all real α and β not both 0 there exists a matrix P (which can depend on α and β) such that

$$P^{-1}(\alpha A + \beta B)P = D, \tag{13.2}$$

where D is a real diagonal matrix. Note that as a particular case the one-dimensional subproblems

$$\vec{u}_t = A\vec{u}_x, \quad \vec{u}_t = B\vec{u}_y,$$

are hyperbolic. (13.2) says that if you look for solutions to (13.1) of the form

$$\vec{u}(t, x, y) = \exp(i\omega t)\exp(i\alpha x)\exp(i\beta y)\vec{u}_0,$$

i.e., if you Fourier transform in both spatial directions, then $\vec{u}_0$ must be an eigenvector of $\alpha A + \beta B$ and ω, the corresponding eigenvalue, is real.

A special case occurs when A and B are symmetric. In this case the matrix $\alpha A + \beta B$ is symmetric and (13.2) always holds. Such a system is called a symmetric hyperbolic system.

While symmetric hyperbolic systems are common, there is one very major complication in going from 1D to 2D. Generally A and B cannot be simultaneously diagonalized. In particular for matrices with a complete set of eigenvectors if A and B are simultaneously diagonalizable then they must commute, which is generally not the case. In order to see what this implies, note that for the 1D system

$$\vec{u}_t = A\vec{u}_x, \tag{13.3}$$

you can transform the equation to diagonal form and for most purposes (e.g., von Neumann analysis but not boundary conditions) you only have to consider the scalar

problem. Unfortunately for 2D problems you generally cannot do this. You can make A diagonal by a similarity transformation, but then the transformed B will not be diagonal. Thus in general you cannot just study the scalar problem in 2D and then jump to study systems.

Note that we found a similar problem when we studied initial boundary value problems in 1D. Such problems cannot generally be decoupled because all variables are coupled by the boundary condition. If you remember how much more difficult the theory was for initial boundary value problems than for just initial value problems you can understand that this is a big complication.

Example, 2D Leap Frog

As you saw above, the problem of going from a scalar equation to a system is complicated in 2D. Even for scalar equations there are some important differences between finite difference schemes in 1D and in 2D. As an example consider the scalar equation

$$u_t = u_x + u_y, \tag{13.4}$$

and the leap frog scheme

$$v_{j,k}^{n+1} = v_{j,k}^{n-1} + \frac{\Delta t}{\Delta x}(v_{j+1,k}^n - v_{j-1,k}^n) + \frac{\Delta t}{\Delta y}(v_{j,k+1}^n - v_{j,k-1}^n), \tag{13.5}$$

where we use the index j to index the $x-$points and the index k to index the $y-$points.

Note that there is no reason that Δx should equal Δy. However, for simplicity we will take $\Delta x = \Delta y = h$. To do a von Neumann analysis in 2D, plug a solution of the form

$$v_{j,k}^n = z^n \exp(i\alpha hj) \exp(i\beta hk) \tag{13.6}$$

into (13.5) to get an equation for z. After some simple algebra you can see that z satisfies

$$z^2 - 1 = 2i\lambda z(\sin(\alpha h) + \sin(\beta h)). \tag{13.7}$$

Note that α and β are independent. For simplicity of notation set

$$\theta = \alpha h, \quad \phi = \beta h,$$

where

$$| \theta |, \quad | \phi | \leq \pi.$$

(13.7) can now be rewritten as

$$z^2 - 1 = 2i\lambda z(\sin(\theta) + \sin(\phi)).$$

Now we see that the von Neumann analysis is more complicated in 2D because we have to consider two Fourier variables (θ and ϕ).

We can solve for z,

$$z = i\lambda(\sin(\theta) + \sin(\phi)) \pm \sqrt{1 - \lambda^2(\sin(\theta) + \sin(\phi))^2}, \tag{13.8}$$

and we will have $\mid z \mid \leq 1$ provided

$$\lambda^2(\sin(\theta) + \sin(\phi))^2 \leq 1. \tag{13.9}$$

(In this case $\mid z \mid = 1$). In the 1D case this requires $\lambda \leq 1$, but in the 2D case the maximum of $\sin(\theta) + \sin(\phi)$ is 2. Thus the stability bound for the leap frog in 2 dimensions is

$$\lambda \leq \frac{1}{2}. \tag{13.10}$$

This indicates an important point. The effect of running leap frog in 2D is to reduce the maximum allowed timestep. This is bad for both efficiency (you have to run at smaller timesteps) and accuracy (leap frog works best for λ near 1).

This behavior is general. Most 2D schemes for the scalar case have a reduced timestep over the 1D version. The way around this is to use operator splitting which we will discuss soon. Note that there will be an additional reduction in going from 2D to 3D.

What can you say about a system? Consider the 1D case,

$$\vec{u}_t = Au_x.$$

By what we did before we found that the stability bound was controlled by the largest eigenvalue of A, $\lambda \leq h/\mu_{max}$. Furthermore the system could be decoupled to look like m scalar equations.

In 2D the situation is more complicated. As before take $\Delta x = \Delta y = h$. If we write leap frog for the system (13.1) and plug in the vector

$$\vec{v}^n_{j,k} = z^n \exp(i\alpha hj) \exp(i\beta hk)\vec{v}_0,$$

we get the system of equations,

$$((z^2 - 1)I - 2i\lambda z(A\sin(\theta) + B\sin(\phi)))\vec{v}_0 = 0. \tag{13.11}$$

If we call the matrix on the left in (13.11) $G(z, \theta, \phi, A, B)$ then the stability condition is that for every θ, ϕ there exist no values z with $\mid z \mid > 1$ such that

$$\det\, G(z, \theta, \phi, A.B) = 0. \tag{13.12}$$

(13.12) is much harder to verify than for the scalar case. In particular, you cannot reduce it to a bunch of scalar equations unless A and B are simultaneously diagonalizable, i.e., they commute, which is generally not the case. What happens in practice is that the stability bound depends not only on some measure of the size of A and B but on how far they are from commuting.

Let's examine (13.11) and (13.12) in more detail. For every value of θ and ϕ, the matrix $A\sin(\theta)+B\sin(\phi)$ is similar to a real, diagonal matrix. This is the definition of hyperbolicty. Clearly you will get a null vector of G if and only if $\vec{v}_0$ is an eigenvector of $A\sin(\theta)+B\sin(\phi)$. For every value of θ and ϕ let $\mu_m(\theta,\phi)$ be the largest eigenvalue (in magnitude) of $A\sin(\theta)+B\sin(\phi)$. Now what you can say is that for every θ and ϕ the problem of determining z reduces to the scalar case and we have the stability bound

$$\lambda \mid \mu_m(\theta,\phi) \mid \leq 1. \tag{13.13}$$

Now if $A = B$, so that A and B are certainly simultaneously diagonalizable, (13.13) reduces to

$$\lambda \mid \mu_{max} \mid \leq \frac{1}{2},$$

where μ_{max} is the largest eigenvalue (in magnitude) of A.

If A and B are symmetric and commute, so that they have a common set of eigenvectors, then we get

$$\lambda \max_l(\mid \mu_l^A \sin(\theta) + \mu_l^B \sin(\phi)) \mid \leq 1$$

where μ_l^A and μ_l^B stand for the eigenvalues of A and B respectively associated with the l^{th} eigenvector ($l = 1, \cdots, m$). The stability bound is now

$$\lambda \max_l(\mid \mu_l^A \mid + \mid \mu_l^B) \mid \leq 1. \tag{13.14}$$

For the commuting cases at least you only have m cases to consider. If A and B do not commute you cannot even say this. Clearly from matrix norms you have

$$\mid \mu_m(\theta,\phi) \mid \leq \parallel A \parallel + \parallel B \parallel,$$

for any matrix norm. You will then have stability provided

$$\lambda(\parallel A \parallel + \parallel B \parallel) \leq 1, \tag{13.15}$$

however (13.15) is generally a severe underestimate. Remember that there can be many matrix norms so you would at least like to use the smallest matrix norm. For symmetric matrices you get the best bound by using the $\parallel \parallel_2$ norm, which you should know from linear algebra is just the largest eigenvalue (in magnitude) of the matrix. If we denote this by $\rho(A)$ and similarly for B we have

$$\lambda(\rho(A) + \rho(B)) \leq 1.$$

This is not a sharp bound (i.e., you can get stability for higher values of λ). In general this bound is worse than (13.14) which is valid when the matrices commute. Note, $\rho(A)$ is called the spectral radius of A.

To summarize, even for the simple leap frog scheme, stability for 2D schemes is very tricky. It depends on more than the eigenvalues when the matrices can not be simultaneously diagonalized (i.e., they do not commute). The situation for 2D Lax Wendroff, MacCormack and Runge Kutta is similar. We will not go further into stability for 2D schemes. In general the bound (13.15) is the best you can do without taking into account specific properties of the matrices A and B. We will discuss a way around these problems called operator splitting.

Operator Splitting

It is possible to reduce the 2D problem to a sequence of 1D problems and then apply just 1D schemes. This technique is called operator splitting and is useful in other contexts as well. In order to motivate this we consider the forward Euler scheme (which is unstable) for the scalar equation

$$u_t = u_x + u_y. \tag{13.16}$$

The scheme is

$$v_{j,k}^{n+1} = v_{j,k}^n + \frac{\lambda}{2}(v_{j+1,k}^n - v_{j-1,k}^n) + \frac{\lambda}{2}(v_{j,k+1}^n - v_{j,k-1}^n), \tag{13.17}$$

where we have assumed that $\Delta x = \Delta y = h$. Remember that this scheme is unstable. It is being used only for teaching purposes.

Now we look for solutions of (13.17). Since we are not doing a von Neumann analysis we will not use z but rather we set

$$v_{j,k}^n = a^n \exp(i\theta j) \exp(i\phi k),$$

and plug into (13.17). We get

$$a^{n+1} = a^n(1 + i\lambda \sin(\theta) + i\frac{\lambda}{2}\sin(\phi)). \tag{13.18}$$

Now for any scheme the only waves that you are approximating accurately are those with θ and ϕ small. From basic calculus if ϵ is small we have

$$1 + \epsilon \simeq \exp(\epsilon). \tag{13.19}$$

Thus for θ and ϕ small we have

$$1 + i\lambda(\sin(\theta) + \sin(\phi)) \simeq \exp(i\lambda(\sin(\theta) + \sin(\phi))$$

and

$$\exp(i\lambda(\sin(\theta) + \sin(\phi)) = \exp(i\lambda \sin(\theta)) \exp(i\lambda \sin(\phi)), \tag{13.20}$$

Since θ and ϕ are small, (13.19) and (13.20) imply

$$1 + i\lambda(\sin(\theta) + \sin(\phi)) \simeq (1 + i\lambda \sin(\theta))(1 + i\lambda \sin(\phi)). \tag{13.21}$$

Now $1 + i\lambda \sin(\theta)$ is exactly what you would get if you applied forward Euler to the 1D equation

$$u_t = u_x,$$

and similarly for $1 + i\lambda \sin(\phi)$. Thus the right hand side of (13.21) is exactly what you would get from applying the split scheme of

$$\tilde{v}_{j,k} = v^n_{j,k} + \frac{\lambda}{2}(v^n_{j+1,k} - v^n_{j-1,k}), \tag{13.22}$$

followed by

$$v^{n+1}_{j,k} = \tilde{v}_{j,k} + \frac{\lambda}{2}(\tilde{v}_{j,k+1} - \tilde{v}_{j,k-1}). \tag{13.23}$$

Note that (13.22) corresponds to solving the 1D equation

$$u_t = u_x$$

for one timestep (since the solution then does not correspond to any physical time we denote them by $\tilde{\cdot}$) and then, using these intermediate values as initial conditions, solving the equation

$$u_t = u_y$$

for one timestep to get the solution at level $n + 1$. We have split the 2D scheme into a product of 1D schemes.

Of course the split scheme is a product of two unstable schemes and is no better than the original unstable scheme. However, the idea can be applied to stable schemes (e.g., MacCormack) as well.

Operator Formulation

Before we apply operator splitting in a more specific way, we consider a general formulation in the context of ordinary differential equations.

First consider the scalar equation

$$\frac{dy}{dt} = (a + b)y, \tag{13.24}$$

where y, a and b are scalars, with a and b given constants. It should be clear to you from very elementary differential equations that the solution to (13.24) at time level t_{n+1} can be obtained from $y(t_n)$ by the formula

$$y(t_{n+1}) = \exp((a + b)\Delta t)y(t_n),$$

where Δt is the timestep.

Using the properties of exponentials we can write

$$\exp((a + b)\Delta t) = \exp(a\Delta t)\exp(b\Delta t),$$

so that

$$y(t_{n+1}) = \exp(a\Delta t)(\exp(b\Delta t)y(t_n)). \tag{13.25}$$

Now you may notice that I have inserted some extra parenthesis in (13.25). This is to emphasize that this equation can be interpreted as first solving the scalar equation

$$\frac{dy}{dt} = by, \tag{13.26}$$

for one timestep and then using this as initial condition (i.e., the solution at time level n) to advance the equation

$$\frac{dy}{dt} = ay, \tag{13.27}$$

one timestep. You should be aware that the update for (13.26) has no meaning in the context of the original problem (13.24). It is simply a numerical device allowing you to replace one equation by two others in the solution process.

This is splitting in its most elementary form. Equations (13.26) and (13.27) are the split equations. Of course, nobody would do this since it is just as easy to solve (13.24) as it is to solve (13.26) and (13.27), however, we will ultimately interpret a and b as differential operators in the x and y directions, respectively. In this case there is a big savings in going from the unsplit to the split equations.

Before we consider differential operators, we next consider the vector system of equations

$$\frac{d\vec{y}}{dt} = (A + B)\vec{y},$$

where $\vec{y}$ is a vector (the size doesn't matter) and A and B are matrices. The solution is

$$\vec{y}(t) = \exp((A + B)t)\vec{y}(0),$$

where the exponential of a matrix is defined by its Taylor series. Now look at what happens if you advance the solution one timestep. The solution will be

$$\exp((A + B)\Delta t)\, \vec{y}(0).$$

Now for a scalar you know the exponential of a sum is the product of the exponentials, i.e.,

$$\exp((a + b)\Delta t) = \exp(a\Delta t)\exp(b\Delta t).$$

You can ask if this is true for matrices as well. The answer is in general no. In fact, from the Taylor series we have

$$\exp((A + B)\Delta t) = I + (A + B)\Delta t + \frac{(A + B)^2\Delta t^2}{2!} + \ldots,$$

and in general,

$$\exp((A + B)\Delta t) \neq \exp(A\Delta t)\exp(B\Delta t).$$

unless A and B commute.

Now suppose you want to do approximations. From the Taylor series you can see that

$$\exp((A+B)\Delta t) = \exp(A\Delta t)\exp(B\Delta t) + \frac{\Delta t^2}{2!}(AB - BA) + O(\Delta t^3). \tag{13.28}$$

This follows from manipulation of the Taylor series. We will not go through it because it is straightforward but tedious. (13.28) says that if you are only interested in first order accuracy in time then you have

$$\exp((A+B)\Delta t) = \exp(A\Delta t)\exp(B\Delta t) + O(\Delta t^2).$$

Note that while this is only first order accurate in general, it is exact if A and B commute.

Since we generally want at least second order accuracy in time the natural question is whether you can do better. In fact you can. You can verify that

$$\exp((A+B)2\Delta t) = \exp(A\Delta t)\exp(B2\Delta t)\exp(A\Delta t) + O(\Delta t^3). \tag{13.29}$$

Using (13.29) (without the $O(\Delta t^3)$) term leads to what is called second order splitting.

You can ask what is the point of this. (13.28) and (13.29) are defined for matrices. From linear algebra you should know that you can also consider a matrix an operator. Consider now the equation

$$\vec{u}_t = A\vec{u}_x + B\vec{u}_y. \tag{13.30}$$

We can write (13.30) as an operator equation

$$\vec{u}_t = L_x\vec{u} + L_y\vec{u}, \tag{13.31}$$

where L_x is the operator $A\partial/\partial x$. and L_y is the operator $B\partial/\partial y$. Now by analogy with the ODE case the general solution to (13.31) can be expressed as the exponential

$$\exp((L_x + L_y)t).$$

(Everything has to be defined very carefully because unlike matrices L_x and L_y are unbounded operators. However, just proceed formally.) We then have the first order splitting to advance one timestep

$$\exp((L_x + L_y)\Delta t)\,\vec{u}(t) \simeq \exp(L_x\Delta t)\exp(L_y\Delta t)\,\vec{u}(t), \tag{13.32}$$

and the second order splitting

$$\exp((L_x + L_y)2\Delta t)\,\vec{u}(t) \simeq \exp(L_x\Delta t)\exp(L_y 2\Delta t)\exp(L_x\Delta t)\,\vec{u}(t). \tag{13.33}$$

Now $\exp(L_x\Delta t)$ is just the solution operator to the 1D equation

$$\vec{u}_t = A\vec{u}_x, \tag{13.34}$$

while $\exp(L_y \Delta t)$ is just the solution operator to the 1D equation

$$\vec{u}_t = B\vec{u}_y. \tag{13.35}$$

Now you should know many ways to solve these 1D equations. Think of MacCormack. Let S_x and S_y stand for solution operators for updating (13.34) and (13.35) respectively one timestep. Then we can approximate the 2D equation (13.30) by

$$v_{j,k}^{n+1} = S_x S_y v_{j,k}^n, \tag{13.36}$$

which corresponds to the first order splitting and by

$$v_{j,k}^{n+2} = S_x S_y S_y S_x v_{j,k}^n, \tag{13.37}$$

which corresponds to the second order splitting.

You might want to think of S_x and S_y as calling a MacCormack subroutine for one timestep. When you do a formal von Neumann analysis (it is very simple because all you are doing are 1D schemes) you can find that stability is governed only by the stability bound for the 1D equations. Note that schemes used with splitting must be only 1 step schemes. Schemes like leap frog can not be split because in the intermediate step of the splitting the solution makes no sense, it is not a solution of the given problem, while leap frog requires data at level $n-1$ as well as at level n.

Note that splitting is not only stable but it is easy to code since you only have to write subroutines for 1D equations. Generally during the splitting you do extrapolation of the fluxes at each boundary and only update the boundary conditions after each timestep has been completed. This formulation is equally valid for the 2-4 MacCormack and can also be applied to Runge-Kutta. Finally, you can use splitting for problems where the timestep for one of the 1D problems say the $y-$problem is smaller than for the $x-$problem. Suppose for example that the $y-$problem required half the timestep of the $x-$problem. For a first order splitting you can do

$$v_{j,k}^{n+1} = S_x(\Delta t) S_y(\frac{\Delta t}{2}) S_y(\frac{\Delta t}{2}) v_{j,k}^n.$$

Note that despite the formal order of accuracy, there is a splitting error so you should be careful when you do splitting. However, in many problems the improvements in efficiencies due to splitting more than overcome additional truncation errors.

You should know that operator splitting need not be applied just to split off x and y derivatives. It can be applied to split off different terms in an equation. For example, suppose you had a diffusion reaction equation,

$$u_t = u_{xx} + R(u). \tag{13.38}$$

Equations of the form (13.38) can be dealt with by semi-implicit schemes as we have already learned. Another approach is to split off the nonlinear term. Thus instead

of advancing (13.38) one whole timestep you can advance the ordinary differential equation

$$\frac{du}{dt} = R(u), \tag{13.39}$$

one time step followed by a solution to the linear partial differential equation

$$u_t = u_{xx}. \tag{13.40}$$

This approach can be thought of as another form of a semi-implicit scheme. It avoids doing a fully nonlinear implicit scheme. Another application would be if the time scales associated with the reaction term were much smaller than the timescales associated with the diffusion terms. Suppose for example that the ODE (13.39) required timesteps 10 times smaller than the PDE (13.40). Then you could solve (13.39) 10 times with the smaller timestep and then use splitting to solve (13.40).

Splitting methods such as these are used in chemistry and nonlinear optics. The main thing to be aware of is that there may be splitting errors which, for practical timesteps, could dominate the overall error.

Approximate Factorization

When operator splitting is applied to an implicit scheme it is called approximate factorization. To see how this works consider first the 1D equation

$$u_t = u_x,$$

(having u_{xx} on the right hand side would not change anything). Suppose we use backward Euler (Crank-Nicolson would be similar) together with the δ-formulation. We would get the linear, tridiagonal system of equations

$$\delta_j - \frac{\lambda}{2}\delta_{j+1} + \frac{\lambda}{2}\delta_{j-1} = \Delta t \frac{(u^n_{j+1} - u^n_{j-1})}{2h}. \tag{13.41}$$

In practice the linear system (13.41) is supplemented by boundary conditions (both imposed and numerical) at say $j = 0$ and $j = N$. Generally the boundary conditions preserve the tridiagonal structure of (13.41). Tridiagonal systems of equations can be easily solved and the computational cost of the implicit scheme is not much more than that of an explicit scheme. A similar statement is true for systems of equations,

$$\vec{u}_t = A\vec{u}_x,$$

where $\vec{u}$ is an m-vector and A is an $m \times m$ matrix. In this case you get a block tridiagonal matrix with blocks of size $m \times m$. The system can be solved efficiently, however the cost of inverting the blocks scales as m^3 so the cost of the implicit scheme increases significantly as m increases.

The situation in 2D is worse. Consider the 2D equation

$$u_t = u_x + u_y,$$

and the natural extension of backward Euler to 2D. Using the δ-formulation and assuming $\Delta x = \Delta y = h$ we have

$$\delta_{j,k} - \frac{\lambda}{2}(\delta_{j+1,k} + \delta_{j-1,k} - \delta_{j,k-1} + \delta_{j,k+1}) = \Delta t(\frac{(u^n_{j+1,k} - u^n_{j-1,k})}{2h} + \frac{(u^n_{j,k+1} - u^n_{j,k-1})}{2h}). \tag{13.42}$$

Now (13.42) is a linear system, but it is no longer tridiagonal. If there are N points in each direction the associated matrix has bandwidth N. In order to see this, note that if you were to assemble the system (13.42) as a matrix, you would have to order points by a single index. Suppose you ordered as in Fortran by letting the first index increase first. Thus the two dimensional array would be ordered

$$(1,1),(2,1),(3,1),\ldots,(N,1),(1,1),(1,2),(1,3),\ldots.$$

In this case the associated matrix would have elements on the main diagonal (corresponding to the elements (j,k) in the equation), elements on the first sub and super diagonal (corresponding to the elements $(j-1,k)$ and $(j+1,k)$) and nonzeros on the N^{th} diagonals (corresponding to the elements $(j,k+1)$ and $(j,k-1)$). Now these kinds of systems (called banded systems) are much more difficult to solve than tridiagonal systems. Not only must you do more computational work, but you must allocate storage for the intermediate diagonals. As a result, direct application of the 2D scheme based on solving (13.42) could be prohibitively expensive. The situation is worse in 3D.

One possible solution is to use an iterative solution method for the linear system (13.42). For example, you may have heard of Conjugate Gradient methods, the Gauss-Seidel method, or the SOR method. However, these methods could require a lot of iterations to converge. Another approach is to split the x terms and the y terms in the matrix, a process called approximate factorization. To see how this works, let's simplify the writing by introducing the notation D_x and D_y to denote the central difference approximations in x and y respectively. (13.42) can then be rewritten as

$$(I - \Delta t D_x - \Delta t D_y)\vec{\delta} = rhs, \tag{13.43}$$

where at this point we do not care exactly what the right hand side of (13.43) is. We then follow the formalism of splitting by introducing the first order approximation

$$(I - \Delta t D_x - \Delta t D_y) \simeq (I - \Delta t D_x)(I - \Delta t D_y). \tag{13.44}$$

The approximate factorization (13.44) reduces the 2D problem (13.42) to the product of two one dimensional operators which can be represented by tridiagonal matrices. Of course you will have to reorder the points when you switch from the solution of the x

system to the solution of the y system. However, this is just an issue of programming. You may have heard of the Alternating Direction Implicit (ADI) method. We will not give the details, but this is very similar to approximate factorization as we have described here.

The approximately factored system is explicitly

$$(I - \Delta t D_x)(I - \Delta t D_y)\vec{\delta} = \Delta t(D_x + D_y)\vec{u}^n. \tag{13.45}$$

Note that when you have steady state conditions, i.e., when $\vec{\delta} = 0$, you still solve the finite difference approximation to the steady state equations (right hand side of (13.45)=0). However, the splitting adds a splitting error to the original discretization error. The splitting error grows as Δt^2 which is bad since you want to use large timesteps with your implicit scheme. Thus you are forced to reduce your timestep if you are interested in the transient solution, however the equations are much easier and cheaper to solve at each timestep.

Nonlinear Equations - Linearization

Things get worse when the equation is nonlinear. Consider first the 1D conservation law,

$$u_t = f_x, \tag{13.46}$$

where $f(u)$ is a given nonlinear flux function. Using the notation f_j^{n+1} for $f(u_j^{n+1})$ we can write backward Euler as

$$u_j^{n+1} - \frac{\lambda}{2}(f_{j+1}^{n+1} - f_{j-1}^{n+1}) = u_j^n. \tag{13.47}$$

Thus at every timestep we get a system of nonlinear equations. It is not even clear how to extend the δ-formulation.

Suppose we set $\delta_j = u_j^{n+1} - u_j^n$ and rewrite (13.47) as

$$\delta_j - \frac{\lambda}{2}(f(u_{j+1}^n + \delta_{j+1}) - f(u_{j+1}^n)) + \frac{\lambda}{2}(f(u_{j-1}^n + \delta_{j-1}) - f(u_{j-1}^n)) = \frac{\lambda}{2}(f_{j+1}^n - f_{j-1}^n). \tag{13.48}$$

In this case we would still have a system of nonlinear equations. In order to get a system of linear equations, assume that δ_j will be small compared to u_j^n and linearize (13.48). For example, we can linearize the first difference involving f in (13.48) by

$$f(u_{j+1}^n + \delta_{j+1}) - f(u_{j+1}^n) \simeq f'(u_{j+1}^n)\delta_{j+1}.$$

The linearized system of equations becomes

$$\delta_j - \frac{\lambda}{2}f'(u_{j+1}^n)\delta_{j+1} + \frac{\lambda}{2}f'(u_{j-1}^n)\delta_{j-1} = \frac{\lambda}{2}(f_{j+1}^n - f_{j-1}^n). \tag{13.49}$$

We now have a linear system of equations for $\vec{\delta}$ with the nice property that at steady state the solution solves the finite approximation to the steady state PDE. One disadvantage of this approach is that the tridiagonal system must be reformed at each

timestep because the coefficients depend on n, however this is more of a programming problem than a computational problem.

Now we can reformulate (13.49) to relate it to something that you may be more familiar with. Suppose you get rid of λ in (13.49), i.e., write

$$\lambda = \frac{\Delta t}{h},$$

and explicitly divide by Δt. Suppose you then consider what happens when $\Delta t \to \infty$. You can see that in this case you will get exactly Newton's method for the nonlinear steady state equation

$$f_x = 0. \tag{13.50}$$

This is both good and bad.

Sometimes you are solving the time dependent equation only to get to the steady state, i.e., the solution to (13.50). In this case you are not interested in accuracy for the transient, you want to use large timesteps, but more importantly you want to get to steady state with the fewest number of timesteps. If your initial data is close to the steady state solution you know that Newton's method converges very rapidly to the solution, i.e., a small number of iterations. This is good. Note that looking at the problem this way, the timestep has no meaning. It is just another way of counting the iterations. When you use large timesteps you can not claim any accuracy on the transient. It is just an iterative method to solve (13.50).

On the other hand you should know that Newton's method tends to be very sensitive to the initial conditions and if you don't have a good initial condition the method may diverge and even blow up. This says that the scheme (13.48) need not be convergent for large timesteps and will generally not be unconditionally stable. Thus linearization will generally introduce stability problems. This is bad.

If you have an $m \times m$ system, linearization involves computing the Jacobian matrix. For 2D problems you would often use linearization combined with approximate factorization. Nonlinear 2D systems are very non trivial numerical problems and there are other approaches, which we will not go into, including both perturbations of what we have described above as well as radically different methods such as multigrid methods.

Anisotropy

We have previously analyzed two basic sources of numerical errors, dissipation and dispersion. In addition to these errors there is an additional source of errors in two dimensions. This is anisotropy, which is due to the fact that waves traveling in different directions can have different errors.

To see how this works consider the equation,

$$u_t = u_x + u_y, \tag{13.51}$$

and suppose you discretize only in space,

$$\frac{dv_{j,k}}{dt} = \frac{v_{j+1,k}(t) - v_{j-1,k}(t)}{2h} + \frac{v_{j,k+1}(t) - v_{j,k-1}(t)}{2h} \tag{13.52}$$

In (13.52) we have assumed that the $x-$spacing is equal to the $y-$spacing for simplicity.

Now assume that we have a wave traveling in the direction α. This corresponds to a spatial dependence of the form $\exp(i \mid k \mid (\cos(\alpha)x + \sin(\alpha)y)$. Plug in a wave of the form

$$u(t,x) = \exp(i\omega t)\exp(i \mid k \mid (\cos(\alpha)x + \sin(\alpha)y),$$

into (13.51). It is easy to see that

$$\omega = \mid k \mid (\cos(\alpha) + \sin(\alpha)). \tag{13.53}$$

(13.53) describes the analytic dispersion relation for the two dimensional partial differential equation (13.51). Note that ω strongly depends on the direction in which the wave is traveling (α).

Now consider what happens for the numerical approximation (13.52). Set

$$v(t) = \exp(i\omega_h t)\exp(i \mid k \mid (\cos(\alpha)x + \sin(\alpha)y),$$

where ω_h now comes from the dispersion relation of the difference approximation (13.52). For simplicity of notation set $k_x = \mid k \mid \cos(\alpha)$ and $k_y = \mid k \mid \sin(\alpha)$, i.e., the wave numbers in the $x-$ and $y-$ directions respectively. By what we have done many times before we know that central differences in the $x-$direction correspond to $2i\sin(\mid k \mid \cos(\alpha)h)$ and similarly for the y direction. We then have

$$\omega_h = \frac{\sin(k_x h)}{h} + \frac{\sin(k_y h)}{h}. \tag{13.54}$$

Note that both ω and ω_h depend on α.

Now exactly as we did in the 1D case we can compute the phase error,

$$\omega - \omega_h = k_x(1 - \frac{\sin(k_x h)}{k_x h}) + k_y(1 - \frac{\sin(k_y h)}{k_y h}). \tag{13.55}$$

Since we are only interested in waves which are small we can expand (13.55) for small values of $k_x h$ and $k_y h$. We now replace k_x by $\mid k \mid \cos(\alpha)$ and k_y by $\mid k \mid \sin(\alpha)$ to get

$$\omega - \omega_h \simeq \frac{\mid k \mid^3}{6}(\cos(\alpha)^3 + \sin(\alpha)^3) \tag{13.56}$$

(13.56) is a measure of anisotropy. It says that the error depends on the direction of the wave (α). The angular dependence is given by

$$f(\alpha) = \cos(\alpha)^3 + \sin(\alpha)^3. \tag{13.57}$$

Note that the angular dependence of the error is different from the angular dependence of ω (i.e., of the exact solution). This is an effect of numerical anisotropy. Waves traveling in different directions have different errors. It is easy to see that (13.57) is maximized for $\alpha = 0$ and $\alpha = \pi/2$. Thus waves that travel along the grid have the worst errors. (13.57) is minimized for $\alpha = \pi/4$.

Note that for a difference scheme, e.g., leap frog, MacCormack or split MacCormack, it is necessary to work out the dispersion relation for the z^s that you get from the von Neumann analysis. The analysis can be very complicated. The important thing for you to remember is that the numerical error will in general depend on the direction of wave propagation (anisotropy) and the exact dependence will differ from scheme to scheme. Often (but not always) waves that propagate along the grid have the worst errors. Note that for a wave propagating in the x direction, the wave number in the $x-$direction is $\mid k \mid$, while for a wave number propagating at 45° the wave number in the $x-$direction is $\mid k \mid \sqrt{2}/2$. Thus the $x-$ (and $y-$) wave numbers are reduced for waves traveling skew to the grid, explaining why you generally expect smaller errors for such waves.

Boundary Conditions in 2 Dimensions

As a final 2D topic we consider boundary conditions. Consider the system

$$\vec{u}_t = A\vec{u}_x + B\vec{u}_y, \tag{13.58}$$

where A and B are $m \times m$ matrices. As part of the definition of hyperbolicity it follows that A and B have m real and distinct eigenvalues. Suppose you are working in a rectangular domain. The procedure to determine the number and type of boundary conditions is to neglect tangential derivatives and look only at the equation for the normal derivatives. Thus if, for example, one of your boundaries is the line $x = 0$ you should neglect the $y-$derivatives and determine the number and type of boundary conditions only from the equation

$$\vec{u}_t = A\vec{u}_x.$$

Do the same thing for the numerical treatment, i.e., the characteristic boundary treatment. Thus the general rule, neglect the transverse derivatives and consider only the normal derivatives.

There is no general procedure for corners. They can give you problems. One approach is to impose both the boundary conditions that you would get from the $x-$boundaries and the boundary conditions that you would get from the $y-$boundaries. Expect inaccuracies and oscillations near corners. In some cases you can get instabilities. In this case use of a dissipative scheme can help.